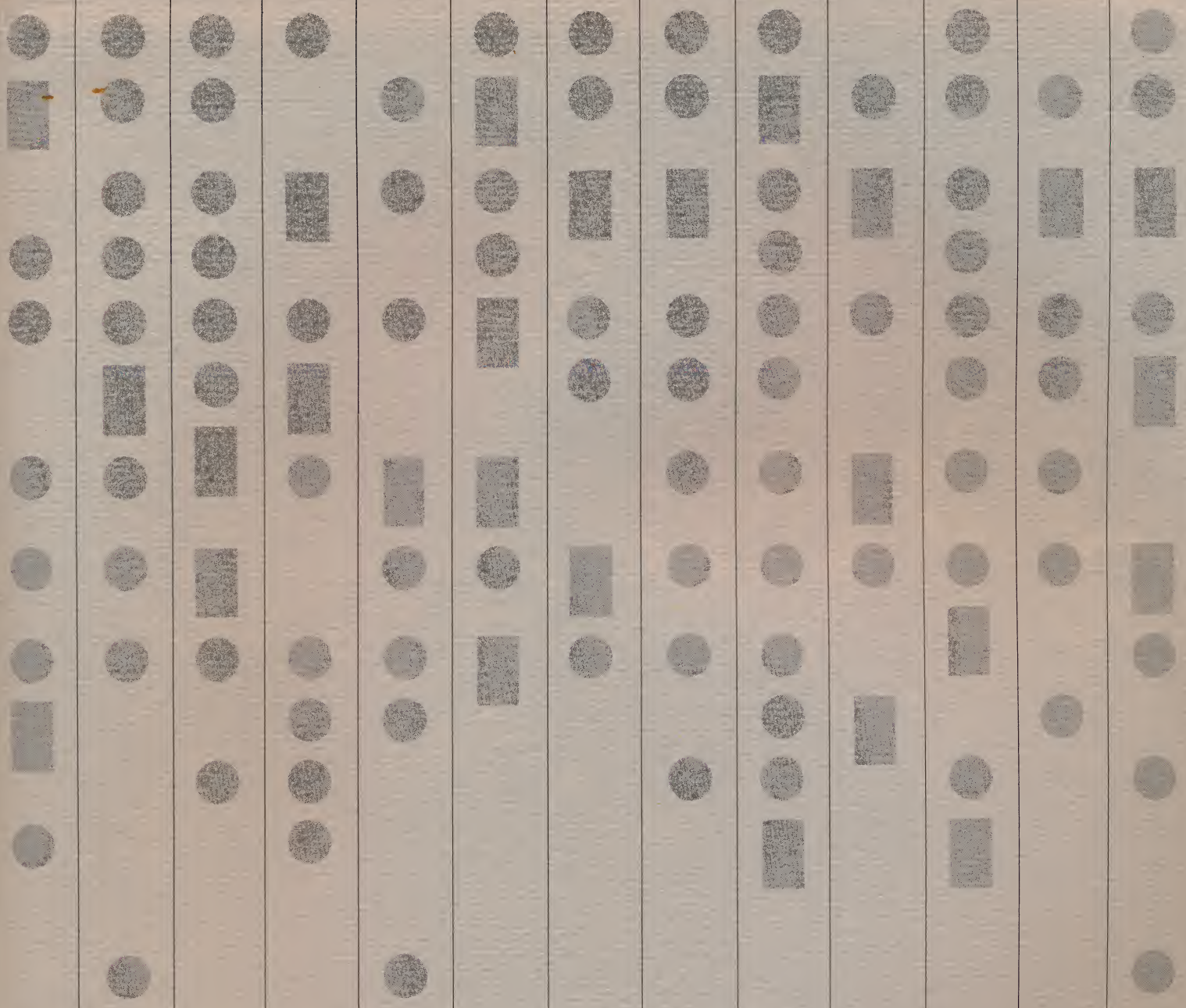
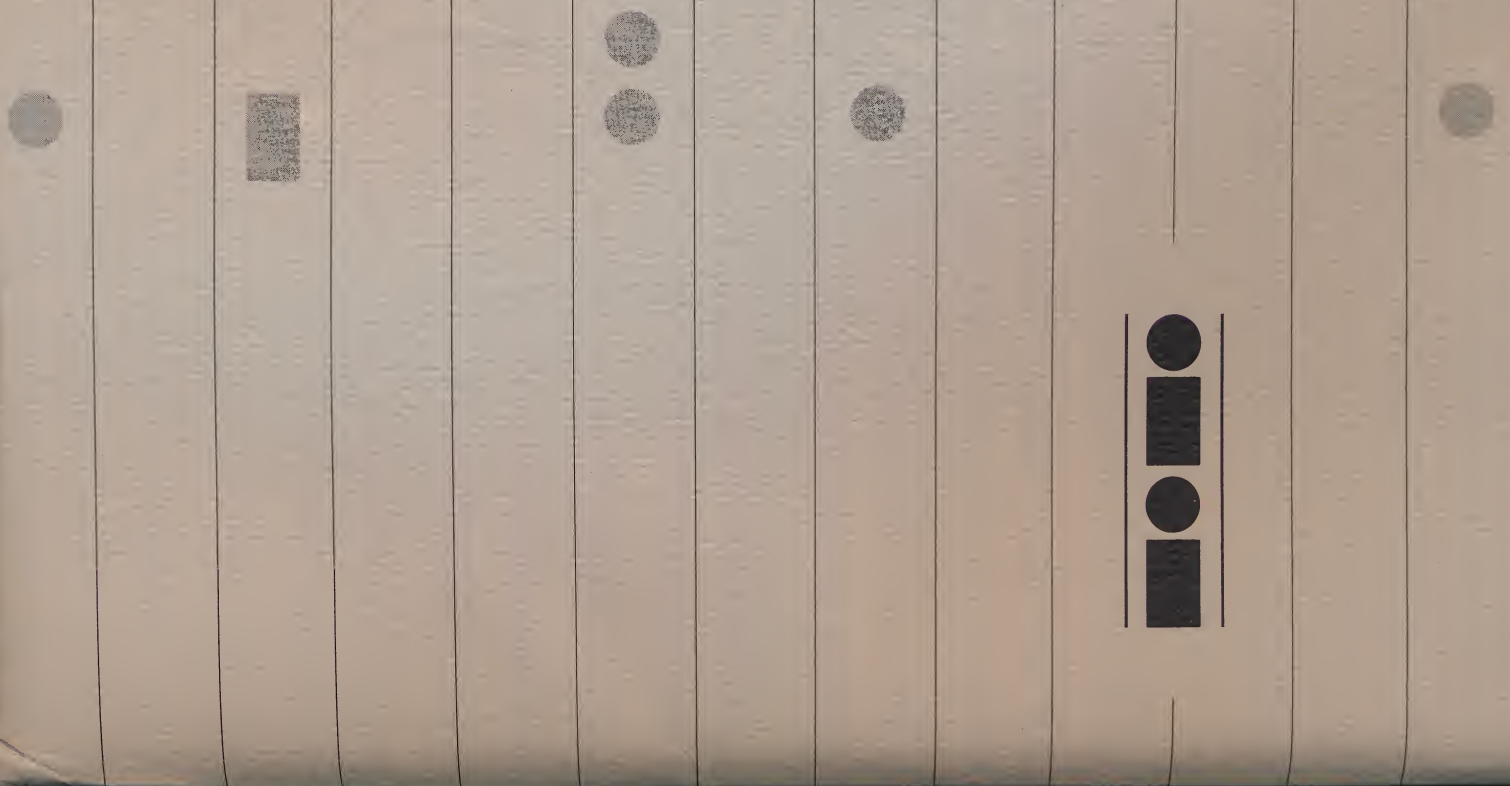




informatics inc.®



**We can give you 300 good reasons why you should use
outside software services. Here are the first eleven.**

J. A. Postley
Advanced
Information
Systems

R. H. Hill
Programming
Systems

F. W. Wagner
Western
Operations

R. C. Lemons
Washington D. C.
Region

W. L. Frank
Eastern
Operations

R. W. Rector
Plans and
Programs

G. J. Vosatka
Marketing

I. Cohen
Systems
Engineering

L. W. Jones
Administration
and Finance

R. D. Archibald
PERT and GPM

W. F. Bauer
President

These men specialize in
the design and
implementation of
information systems,
with primary emphasis
on computer-
communications, file
management, on-line and

WHO WE ARE — WHAT WE DO

Informatics Inc. is a company of specialists. We specialize in helping users of information processing systems get the most out of their equipment. We provide much more than just "computer software." Essentially we are set up to supplement our clients' staffs in areas where a specialized or concentrated effort is necessary. Specifically we offer system analysis and design, programming, technical communications, and specific application services. We can take on a complete project, or just give good advice. In either case we have some of the industry's top specialists to do the job.

Our greatest capability, in terms of experience, is in the area of on-line, real-time systems. We have worked on many projects in this rapidly expanding field, both for government agencies (on command and control, missile tracking and similar projects) and private industry (airline reservations, automatic navigation, etc.).

**We can give you 300 good reasons why you should use
outside software services. Here are the first eleven.**

R. H. Hill
Programming
Systems

R. W. Rector
Plans and
Programs

G. J. Vosatka
Marketing

J. A. Postley
Advanced
Information
Systems

F. W. Wagner
Western
Operations

R. C. Lemons
Washington D. C.
Region

W. L. Frank
Eastern
Operations

I. Cohen
Systems
Engineering

L. W. Jones
Administration
and Finance

R. D. Archibald
PERT and GPM

W. F. Bauer
President

These men specialize in the design and implementation of information systems, with primary emphasis on computer-communications, file management, on-line and display-oriented systems. And they manage our other 289 good reasons.

 **informatics inc.®**
5430 Van Nuys Boulevard
Sherman Oaks,
California 91401

Here's the only advice you need to design your own on-line system: don't

Designing on-line systems is tricky. Nevertheless, do-it-yourselfers often try it, taking twice the time to do it wrong at triple the cost of doing it right. Nobody has that sort of money. Or time. That's why you need an expert to put your system on line. Which expert? We have some advice on that subject, too.

by Richard H. Hill



We're about to shoot do-it-yourselfers right out of the on-line systems design business.

SAYS WHO?

First off, our credentials: At Informatics Inc., we think we have every right to speak on the subject of on-line computer systems. Our justification is simple: we've probably done more work in on-line software than any other group in the field. Fact is, we seem to be the only major programming firm anywhere *specializing* in on-line computer systems implementation. And much of our work has been conducted at the furthest extension of the art (National Military Command System, the RADC on-line computing system and the Mobile Wing Reconnaissance Technical Squadron are a few examples). Do we know what we're talking about? We'd better.

THE MISSING LINK: SOFTWARE

At Informatics, we believe that the computing system exists for the user, not *vice versa*. Consequently we are convinced that the directions of the future point inevitably to direct, on-line user/computer communication. But, if you accept this fact, you also have to accept the problems of putting the user and the computer in direct dialog. How do you do this? It's not easy. Nevertheless, a lot of well-meaning users have tried. And a lot of amoebic

monsters have been spawned—so divided and subdivided that any semblance of direct access to the system is lost. That's why the job has to be done by an expert—someone who's had the course in the complexities of on-line programming. Right now, today, all of the equipment and technology exists to put even the most sophisticated system on line. The only missing ingredient is on-line software.

THE WAITING GAME

The essential key to on-line implementation is *time-sharing*. Modern computers—and even those not so modern—are too fast to serve only one person. To make economic sense, the computer must be shared. Segments of the total computing time must be made available to many users. And not all of the users need be humans: regularly scheduled programs can also have their share in on-line systems. For instance, a computer in a medium-sized manufacturing company might service ten or twelve on-line engineering design consoles, concurrently record sales orders and other messages received by teletype from other company offices, and also compute payroll—all on a time-sharing basis. Difficult? Yes. Impossible? No. It can be done by someone who knows how. And knowing how means mastery of a few knowledge areas: Dynamic storage allocation. Interrupt management. Task queuing. Priority level control. Program rollout and rollback. Random access storage management. Time-slicing. Memory protection. And several other odds and ends of programming technology. Knowing how also means experience. Real, practical, working experience. The I've-done-it-before-and-I-know-exactly-how-to-put-the-whole-thing-together-and-make-it-work-type experience. And, on top of this, knowing how means knowing what *not* to do in on-line implementation. Knowing what

not to do is every bit as important as knowing what to do if the system is to work, work right and work under all conditions.

WE'RE READY, ARE YOU?

If you've read this far, chances are you need an on-line system. And at this point you should realize that we think we can design one for you. If you'd like to talk over your own on-line systems design requirements or if you think you're qualified to help us solve other people's problems, our number is (213) 783-7500. Ask for me, Frank Wagner, Walter Bauer or any other members of our staff. We also have literature on our people and capabilities which we will be happy to send you. Address Department E, Informatics Inc., 15300 Ventura Boulevard, Sherman Oaks, California 91403.

informatics inc.®
15300 Ventura Boulevard
Sherman Oaks, California 91403

Please send me your staff-authored article "Implementation Procedure for On-Line Systems"

Name _____

Title _____

Company _____

Address _____

City _____

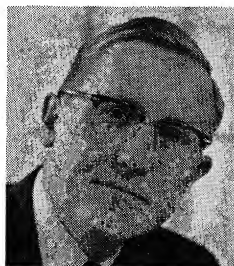
State _____ ZIP _____

An equal opportunity employer

WHY MUST PROGRESS IN COMPUTER USAGE PLUNGE AHEAD AT A SNAIL'S PACE?

Take on-line or real time systems. You'll find the concentration of these systems in defense, space or critical industrial and business applications. In categories less critical than these, more mundane techniques are condoned. And that's too bad. But there's not a lot anyone can do about it in a hurry.

By Dr. Walter F. Bauer



Much has been said about real time computer operation. A great deal of writing has been done about on-line computer systems. Yet, comparatively little has actually been done to integrate the systems into instrumentation and people-loops. This is very curious, since many computers are suitable for such applications, buffering and display equipments exist, and the techniques for implementation are known and understood. The principles are straightforward. The computer is a wonderfully flexible instrument, which can take over much complex system control and even many of the trivial human tasks. Although experience in on-line systems is limited in the industry, the knowledge of software like real time executive control systems is available to those who need it.

WHO'S AT FAULT?

At Informatics Inc., we are particularly well-suited to solve the problems of on-line systems software. We have done a lot of it. In fact, we are the only independent software firm that specializes in solving real time problems and developing on-line computer controlled systems. It follows that if on-line usage has been slow to evolve, we must shoulder a fair share of the blame. We do. And it bothers us, since there is really no logical reason for there being so comparatively few on-line installations. The problem is not that difficult. There are no longer basic impracticalities to the on-line concept. Real time operation is well out of its experimental stages in most areas. Yet much of the progress of data processing is jammed up behind the on-line road block. There are reasons for this. One of these reasons is the need to understand the potential advantages and to plan the system carefully. Another reason lies in our own organization.

WE CAN'T SOLVE OUR OWN PROBLEMS.

Our only reason for existence is to bring efficiency into our customers' operations. That fact necessitates a measure of inefficiency in our own organization. For instance, we have no salesmen. When you talk to a man from

TYPICAL ASSIGNMENTS

Listed here are a few examples of the type of real time software work Informatics is involved in:

PACIFIC MISSILE RANGE: Program design for one of the most complex and critical on-line systems yet developed; tracking, radar target acquisition, data recording, and display involving time requirements twice as strict as currently operational similar applications.

NASA HOUSTON: Programming for mission control, launch abort, orbit, rendezvous and re-entry for Apollo and Gemini missions in the Real Time Computer Complex under development by IBM.

NATIONAL MILITARY COMMAND SYSTEM: Programming and analysis in displays and other on-line aspects for the highest level command and control system in the nation.

UNIVAC: Programming for the Univac computers which automate message switching in the GSA communications system.

OFFICE OF NAVAL RESEARCH: Analysis of the country's hardware and software technology and system planning for Naval Tactical Systems of the 1975 era.

Informatics, you talk to someone who has been directly responsible for the management of successful real time software programs. These are very expensive salesmen. But anyone at a lower level of competence would be unable to evaluate problems properly. Nor could he ask the questions that allow proper evaluation. The techniques are too new. The disciplines still too narrow. The acceptance of on-line concepts still too restricted.

THE SOLUTION: MORE QUALIFIED PEOPLE.

There is a considerable shortage of talent in the general field of software. That is bad enough. But when you further qualify this talent to the specific of on-line systems capability, the shortage is acute. If you canvassed the world for men fully qualified to analyze and solve real time software problems, you'd find them extremely few and far between. We know. We canvass constantly, with the zeal of treasure hunters. We have to. The

nature of our work demands that each and every Informatics staff member be qualified to take on problems of the most modern type, involving computers controlling displays, communication devices, and analog instrumentation. It's not unlike staffing a hospital with nothing but neuro-surgeons. But we can find no alternative. This limits our own growth. It limits the number and nature of the assignments we can accept. And it helps to limit the entire progress of on-line systems usage. But we are making progress. Today we're at work on eight large scale on-line systems!

WHY DID WE PRINT THIS MESSAGE?

This advertisement is appearing for three reasons. First, of course, we hope it puts us in contact with people who have software problems that require our level of capability. Second, we'd like to talk with talented people who are qualified to join us. Third, we hope to concentrate more attention on the entire field of on-line systems. From our point of view, even though the swing to optimizing computer usage must, by its very nature, be evolutionary, everything that's published to expose the problems involved will serve to speed things up a bit.

If you would like to discuss our approach to partially or fully on-line systems software, the best way to do so is by telephone. Our number is (213) 783-7500. Ask for Walter Bauer or Frank Wagner, or any of our other non-salesmen. We also have literature on our people and capabilities which we will be happy to send you on request. Address Department E, Informatics Inc., 15300 Ventura Boulevard, Sherman Oaks, California.

informatics inc.®

15300 Ventura Boulevard
Sherman Oaks, California 91403

Please send me your staff-authored articles checked below:

- ☐ Programming On-Line Systems
☐ A Real Time Data Handling System

Name _____

Address _____

Command and Control: Why say 100010110 when you mean 278?

Today's military commander lives with frustrations that would have caused von Clausewitz to turn in his sword. His mission is decision-making. His decisions are based on alternate choices derived from the best possible data. The only possible way he can get this data with the speed and accuracy required is through a complex maze of computer systems which do not even speak his language. Who can solve this problem of man/machine communication? Not the hardware manufacturer. Nor the military. It's a job for an independent interpreter.

By Werner L. Frank



The problem of man vs. machine in military command and control systems communication is a problem of conflicting viewpoints. Too many of the men who are responsible for the development of

highly sophisticated automatic data processing systems seem to believe that the world must inevitably be recreated in their machines' image. Nowhere is this vanity more prevalent than among those who seek to automate the Military. Their contention seems to be that it is far more logical (and consequently very necessary) for the Military to adapt to their machinery than to attempt to adapt their machinery to the Military. Instead of requiring the machine to communicate with the man, they are requiring the man to communicate with the machine. They are wrong.

WHAT MAKES THE MILITARY TICK?

The very root of the military profession is discipline. Without clear-cut, ascending levels of authority, the military establishment would crumble. What has this to do with military data processing systems? Plenty. Because it has much to do with the role of Commander. In crisis, each man and *each piece of equipment* under the Commander's authority must respond directly and immediately to his will, his needs, his methods, his commands. If he is to realize the full advantage of the computer-centered system at his disposal, he must not be required to use it through a frustrating chain of technicians and codes. He and his staff must be able to communicate with the machine directly — in the language they are accustomed to. To expect the Commander to adapt his requirements and decision-making procedures to the convenience of an electronic command and control system is not only foolish but potentially fatal.

WHAT MAKES THE HARDWARE MANUFACTURER TICK?

The men who design computers and computer-oriented hardware are, by their very nature, machine worshippers. This is as it should be. This is what drives the technology forward. But their near-total dedication to the form and function of the data processing system itself is in sympathy with neither

TYPICAL COMMAND AND CONTROL SYSTEM ASSIGNMENTS

Listed here are a few examples of the command and control software projects in which Informatics is engaged:

National Military Command System: programming and analysis in displays and other on-line aspects for the highest level of command and control system in the nation.

Office of Naval Research: two projects involving advanced Naval Tactical Data Systems: (1) system planning and analysis of the country's hardware and software technology for Systems of the 1975 era and (2) development of a design concept for advanced Marine Tactical Data Systems.

Rome Air Development Center: two projects for the Air Force: (1) the development of the programming system for a man/machine system involving an on-line interrogation console and display used in intelligence evaluations and (2) development of an Executive Control Program for a large-scale, multi-computer system for the development of military data processing techniques.

the immediate needs and problems of their users nor the prevailing concept of evolutionary system upgrading. They are not ready to attack the man/machine language problem.

THE BEST SOLUTION: HIRE AN EXPERIENCED INTERPRETER

The only place where maximum value from a military command and control system can be reached lies about half-way between the producers of the hardware and their military customers. Neither of the two is adequately trained or objective enough to go even half of the way. Someone with full comprehension and appreciation of the problems and attitudes of both must stand in the middle. He must have experience with on-line group displays and display consoles and the software techniques that go with them. These include query languages and systems of operational programs responsive to the Commander on *his* terms. There are few firms with such experience. Informatics Inc. is one of them.

WHAT MAKES INFORMATICS INC. TICK?

We at Informatics Inc. represent the user. We work with the manufacturer towards getting more flexible hardware. At the same time, because command and control requires the closest relationship of man to machine, we help the user to employ his present hardware with the least possible deviation from his normal operating procedures.

Our staff is particularly well suited to the analysis and evaluation of command and control communication problems. Twenty members of our senior staff have been directly responsible for the technical direction of successful real time software programs, including some elaborate interrogation-display systems for such applications as intelligence data handling, own-forces evaluation, damage assessment, and weapon assignment. Further, we have developed some basic techniques that allow rapid and efficient tailoring of general-purpose computers and displays to specific military purposes. Informatics, Inc., is the *only* independent software firm specializing in solving the real time problems and developing the on-line computer-centered systems of command and control.

If you would like to discuss our approach to command and control communications with the idea in mind of solving some of your own problems or perhaps of joining us in solving other people's problems, give us a call. The number is (202) 244-7102. Ask for Werner Frank, or any other member of our Washington, D.C., staff. We also have literature on our people and capabilities which we will be happy to send you on request. Address Department A, Informatics Inc., 15300 Ventura Boulevard, Sherman Oaks, California.

● informatics inc.®
● Department A
● 15300 Ventura Boulevard
● Sherman Oaks, California 91403

Please send me your staff-authored article:

Military Command: A Challenge
For Information Processing.

Name _____

Address _____

An equal opportunity employer

In-house reprogramming is simple.

All it takes is time and money and time and money and time and money and time and money and time and money

by Frank Wagner



Converting an installation from one computer to another always looks like the ideal do-it-yourself job. After all, the programs are all written, debugged, and working. The equipment is compatible. And, being

a wise manager, you further eschew trouble by decreeing that the new programs shall be identical copies of the old. The conversion process thus appears to be purely mechanical, and you assign your youngest programmer to it (for reasons of morale and efficiency) with a clear conscience. But don't settle back too far in your easy chair. You may encounter some problems which will have you reaching for the aspirin bottle, and yelling "Why didn't someone tell me?" O.K., we'll tell you.

SELF-INFLICTED TORTURES

You've no doubt already guarded against *undocumented programs*. So, the first real problem you'll have to contend with is the *bad write-up*. The written specifications do not agree with the program, and you find out only when the rewritten routine is checked out with live data. By the time the discrepancies have been corrected, your cost and schedule have slipped by 20%. Or, you may not be so lucky and miss by 50-75%. Then there are *machine-dependent routines*. These are written by "clever" programmers who love to use illegal instructions. Many a channel program depends on the subtle timing considerations of a particular machine. The poor neophyte, assigned to reprogram such a routine, will often wind up wondering just what it is that the program is really supposed to do. More time and money lost. Worse yet, however, are *software-dependent programs*. A relatively simple case to find and fix is exemplified by the "standard" arithmetic routine which gives slightly different results than the one it replaces. Much harder to track down are the instances where all the installation programs have been written to negate the effect of some obscure bug in the overall operating system. Most likely no one could find the error four years ago, so everybody stopped trying, and finally its very existence was forgotten.

PROMISES, PROMISES...

The new equipment presents its own problems. True, the manufacturers sing an enticing song: "Our hardware is compatible... our software is compatible... our manuals are compatible... our salesmen are compatible..." ad nauseum. But there are degrees of compatibility ranging from the exact match to the totally alien. The trick is to understand the differences, anticipate and appraise the problems, and assign to each its proper degree of importance. For instance, just where are the syntax differences between OS/360 F-level COBOL, BOS COBOL, and CDC COBOL 63? And how do they all relate to COBOL 61? Are the differences easy to cope with, or is the job three times what you expect? There are hardware differences too. "Our tapes are compatible" cry the compatible salesmen. But on what level? Character codes? Binary? Only if you ignore parity checks? Or only if you don't try to read or write records of less than three characters? And what about those tapes you got from an outside source, written by write amplifiers with different tolerances? You have learned how to run them on your old equipment. Are the new tape units *all that* compatible? Listen to the promises, but remember the gal who said "My husband and I are perfectly compatible. He likes to stay out late with the boys, and so do I."

ENTER THE TINKERERS

Finally, in spite of all your admonitions to keep the new programs identical to the old, there's always a temptation to make a few (just a few) improvements. The programmer's very soul revolts at the idea of duplicating some obviously poor characteristics. ("It's really easier to change than to copy"). The users, too, will ache to get rid of some real or imagined inconvenience, or to add a bell here and a whistle there. At Informatics we have a standing offer of reward for the first evidence of a truly identical reprogramming job ever done in-house. There's only one sure way to get one: pay a software house a fixed and firm price to produce it. The profit motivation will assure the careful examination of every bell and whistle with a jaundiced eye. And if improvements are made, to keep you a happy customer, they cost you nothing.

DON'T TRY IT IF YOU HAVEN'T KNOCKED IT

Which brings us right to the point: converting an installation is not as simple as it looks. If you are planning a reprogramming job of some size, come to Informatics and talk to us about helping you. (Or, if you feel qualified and ambitious, let's talk about your helping us.) We have the management depth to tackle this kind of problem. Remember the first large-scale reprogramming task ever undertaken? It involved simultaneously converting 18 large scientific computing installations from IBM 701 to IBM 704 equipment. Many policy decisions made today still go back to the lessons learned from that experience. Some important influences in the computing field grew out of it: SHARE programming standards, incentives for FORTRAN and COBOL, and monitor systems requiring separated input/output systems. Most of our staff members are veterans of that effort and have, additionally, gone the route as managers of their own installations. You recognize the names: Bauer, Wagner, Frank, Hill, Rector, Postley, Cohen, Bigelow, Bright, Lemons, Corbin, Kaylor, Mersel, Mallet, Stofko. If they can't help you, there's only one other place to turn: go see a good psychiatrist. He won't solve your problem, but he'll make you think you like it.

If you would like a copy of Frank Wagner's new paper "Design of a General Purpose Scientific Computing Facility", just print your

Name _____

Company _____

Street _____

City _____

State _____ Zip _____

and return to:

 **informatics inc.®**
5430 Van Nuys Boulevard
Sherman Oaks, California 91401

An equal opportunity employer

**THE DISPLAY ORIENTED
COMPUTER USAGE SYSTEM:
A GENERAL PURPOSE
SOFTWARE PACKAGE
FOR DISPLAY SYSTEMS**

informatics inc.

THE DISPLAY ORIENTED COMPUTER USAGE SYSTEM:
A GENERAL PURPOSE SOFTWARE PACKAGE FOR DISPLAY SYSTEMS

by

Werner L. Frank
Vice President
Informatics Inc.

[This work is being sponsored by the U. S. Air Force,
Rome Air Development Center,
under Contract AF 30(602)3500.]

Presented to the
Eleventh Technical Meeting of the AGARD Avionics Panel
Symposium on Displays for Command and Control Centers
November 7-10, 1966

INFORMATICS INC.

5430 Van Nuys Boulevard
Sherman Oaks, California
USA
(213)783-7500

4720 Montgomery Lane
Bethesda, Maryland
USA
(301)654-9190

ABSTRACT

The increasing utilization of display systems for military Command and Control is leading to the development of a new class of software appropriate to on-line display/computer systems.

The Display Oriented Computer Usage System (DOCUS), under development for the U. S. Air Force at the Rome Air Development Center, is such an example of a complete on-line display software package which has the objective of providing:

- 1) A general purpose operating system for managing on-line display consoles in a multi-access environment.
- 2) A Display Oriented Language which is used to describe and implement man/machine procedures.
- 3) A compiler which accepts the higher order Display Oriented Language and produces conversation mode procedures.
- 4) A library capability for handling of displays.
- 5) A user capability for combining existing operations and displays to generate more complicated operations.

THE DISPLAY ORIENTED COMPUTER USAGE SYSTEM:

A GENERAL PURPOSE SOFTWARE PACKAGE

SLIDE 1

FOR

DISPLAY SYSTEMS

1.0 INTRODUCTION

Military Command and Control is rapidly becoming identified as a complex of communications and computers interacting with man through use of display terminals. It is significant to note that most of the United States Forces command and service centers are rapidly becoming equipped with display console capability.

The presence of the display console in the command and control environment was originally motivated by the desire to insert queries and obtain responses from an established data base. The general utility of those consoles has expanded and a wider class of problems are now solved with them as shown on Slide 2. These include data entry, text manipulation and editing, file manipulation, graphical presentations and the solution of analytical problems such as those found in statistical analysis or engineering computation. In addition, consoles are now being used in generating computer programs, checking them out by on-line debugging, maintaining the

SLIDE 2

programming system, operating system diagnostics for hardware maintenance purposes, and in monitoring and controlling total system operation.

These general application areas, first motivated by the military, are now having a profound effect on the commercial and industrial data processing market as well.

This acceptance of display devices has exerted pressures towards the development of an on-line software technology in support of these terminals. This requirement is much in line with earlier software developments for batch or off-line processing as shown in Slide 3, which led to the building of general purpose programming languages including assemblers and compilers, as well as operating systems such as job monitors and input/output systems. In the on-line environment, however, there is a need for the additional software features of a real time and/or time sharing executive, a communications handler and terminal or display oriented software. It is with respect to the latter subject that the Display Oriented Computer Usage System (DOCUS) is being developed.

SLIDE 3

**THE DISPLAY ORIENTED
COMPUTER USAGE SYSTEM:
A GENERAL PURPOSE
SOFTWARE PACKAGE
FOR DISPLAY SYSTEMS**

informatics inc.

THE DISPLAY ORIENTED COMPUTER USAGE SYSTEM:
A GENERAL PURPOSE SOFTWARE PACKAGE FOR DISPLAY SYSTEMS

by

Werner L. Frank
Vice President
Informatics Inc.

[This work is being sponsored by the U. S. Air Force,
Rome Air Development Center,
under Contract AF 30(602)3500.]

Presented to the
Eleventh Technical Meeting of the AGARD Avionics Panel
Symposium on Displays for Command and Control Centers
November 7-10, 1966

INFORMATICS INC.

5430 Van Nuys Boulevard
Sherman Oaks, California
USA
(213)783-7500

4720 Montgomery Lane
Bethesda, Maryland
USA
(301)654-9190

ABSTRACT

The increasing utilization of display systems for military Command and Control is leading to the development of a new class of software appropriate to on-line display/computer systems.

The Display Oriented Computer Usage System (DOCUS), under development for the U. S. Air Force at the Rome Air Development Center, is such an example of a complete on-line display software package which has the objective of providing:

- 1) A general purpose operating system for managing on-line display consoles in a multi-access environment.
- 2) A Display Oriented Language which is used to describe and implement man/machine procedures.
- 3) A compiler which accepts the higher order Display Oriented Language and produces conversation mode procedures.
- 4) A library capability for handling of displays.
- 5) A user capability for combining existing operations and displays to generate more complicated operations.

2.0 NATURE OF DISPLAY CONSOLES

But what are display consoles? Several examples are illustrated in the accompanying slides. The first is a console manufactured by the Bunker Ramo Corporation and the second, a desk top console made by Control Data Corporation.

SLIDE 4&5

An initial objective of DOCUS was to derive principles of operation of display consoles which are independent of particular hardware idiosyncrasies for both the computer and the display devices.

Display devices are as general purpose as computers and the illustration in Slide 6 shows the features of consoles that can be abstracted to form the hardware conceptualization for DOCUS. These include:

SLIDE 6

- o the Display itself, typically a CRT with character and/or line drawing presentation capability.
- o an Alpha/Numeric Keyboard, much like a typewriter input set.

3.0 THE DOCUS STRUCTURE AND FUNCTION

The Display Oriented Computer Usage System is a fundamental software system providing tools for:

- o Implementing man/machine conversational procedures through a display console oriented language.
- o Compiling these languages to computer operating code.
- o Executing this code within a responsive, on-line and multiple user environment.

Also, it provides users the ability to:

- o Communicate with the data processing system by problem oriented operations.
- o Generate on-line, additional operations built upon existing operations.
- o Manipulate displays.

The DOCUS environmental operation is shown in Slide 8.

SLIDE 8

The general process begins with a conversation mode during which a message (or demand) is generated by the terminal operator. As shown, the terminal interface makes the transformation between the specifics of the particular hardware devices and the internal, general purpose notation of DOCUS. Upon completion of the conversation, the completed message is ready for "background" processing. This is the type of processing that proceeds in either a time shared mode or in a separate computer while other conversations - with the same console or others - continue. Ultimately, when the requested task is completed, the results are transmitted back to DOCUS for display.

An example of a task program may be the retrieval and sort capability of a file management system. DOCUS provides the query and graphic output capability through the specification of a man/machine procedure. This fundamental capability is illustrated in Slide 9 where the user's interaction consists of operating on a set of Function Keys, appropriately labeled for a particular application - and collectively called a procedure. A procedure may be for example:

SLIDE 9

- o a query generation capability
- o a format request for data entry
- o engineering computation

- o composition of output
- o text manipulation

The Function Keys associated with a particular procedure are operable or selectable according to whatever logical structure is imbedded in the definition of the procedure. DOCUS permits the statement of what transpires for each selected Function Key and how these selections are to be ordered.

For example, the illustration identifies in green the possible alternate choices of selecting INTELLIGENCE, OPERATIONS, and LOGISTICS after initiating the blue START key. The red keys indicate that subset of Function Keys which become selectable upon prior selection of, say OPERATIONS.

In each case of key selection the user is typically confronted with one of two types of displays - a list display of alternative choices or actions, or the format display requiring specified data input as shown in Slide 10.

SLIDE 10

In Slides 9 and 10, DOCUS recognizes the needs of two classes of system users: the console operator/analyst and the procedure implementor. The analyst is the ultimate user for whom the

hardware/software system exists; he is the problem solver and he is the person who interacts with data and displays. He is least likely to know or care about details of data processing hardware or software since he looks upon the data processing system as a tool in fulfilling his requirements. Thus, he requires an application oriented language rather than any stylized computer language in order to perform his tasks.

The analyst has at his disposal basic building blocks of elementary functions provided by the procedure implementor. Through the Function Keys he may manipulate these functions on-line through DOCUS by combining, modifying, and organizing them to generate more complicated functions for his use. In this process, he employs terminology and operations familiar to him. Hence, the analyst can generate his own personalized processing capability.

This point was illustrated in Slides 9 and 10 where appropriately labeled Function Keys guide the analyst through his selected procedure. In turn, each Function Key selection causes a list or format display to appear in the user's language through which the man/machine communication proceeds.

DOCUS recognizes the procedure implementor as a data processing technician or programmer who understands the analyst's

applications sufficiently so that he can design and implement procedures for use by the analyst. As such, he will also understand computer programming, the processor hardware and the techniques of man/machine communication.

DOCUS provides software tools to enable the procedure implementor to operate more efficiently through a Display Oriented Language (DOL). The Display Oriented Language is used to express application procedures and facilitates their implementation within the DOCUS operating system. The Display Oriented Language consists of declarative and operator statements which govern the step-by-step process of man/machine communication together with the necessary commands for creating and manipulating operational elements, e. g. , displays. The power of DOL is enhanced by including within the language generalized operations consistent with the hardware abstractions already defined and associated with consoles. Hence, DOL programs written for a display console of type A would operate on recompilation with display console of type B. SLIDE 11

In addition to DOL, the procedure implementor has available for on-line use debugging and utility programs.

Slide 12 identifies further features of DOCUS. The

SLIDE 12

Executive Control provides the input/output terminal interface and multi-terminal operation through appropriate time sharing techniques. (The latter may be accomplished by tying DOCUS to the overall system executive if it has time sharing capability). The DOCUS Executive is capable of handling multiple and diverse procedures simultaneously being exercised at the various terminals.

A second important DOCUS function is the control of displays, which are the operands upon which all system functions are performed. Displays can be constructed and manipulated through the user oriented language capability, then preserved in a library from which they can be accessed by symbolic referencing.

Finally, DOCUS provides the all important tool for the implementation of conversation mode procedures. With the Display Oriented Language it is possible to define the man/machine steps that take place at each selection of a Function Key. These DOL expressed steps are called Key Programs. As already noted, a logically connected set of such Key Programs is called a Procedure. DOCUS provides the compiler to translate these DOL written Key Programs into computer code. The DOCUS Executive system accommodates these DOL programs and facilitates interface with the Display Library.

4.0 AN EXAMPLE

For purposes of illustration an example is selected to demonstrate the utility of DOCUS.

The application selected is Message Data Entry (Slide 13) SLIDE 13 where the alternate sources of data are teletype messages, voice (telephone) calls, clearly specified hard copy formatted information and free text. The selection of the key "Message Entry" is to call to the console operator's attention a choice of various data sources as shown in Slide 13. Selection by, say light pen, of the "Teletype" SLIDE 14 alternative leads to the display "Teletype Message Formats" (Slide 14) from which the operator makes a further appropriate selection based SLIDE 15 on the type of message he wants to enter. Having made his selection from this display he will be presented with a structured format display which will prompt him to enter the required data based on the message type. Slide 16 assumes a selection of REDOPS. SLIDE 16

The procedure implementor can generate the appropriately required user dialogue by employing DOCUS in the following manner:

- o Create at the console the displays
MSGSRC, TTYMSG and RDPFMT and place
them in the DOCUS Library. This display
definition process permits the building of

a family of displays wherein each item of a list display points to a possible associated display. Thus, an entire tree structure of displays can be defined reflecting the content, organization and hierarchical structure of a data base. Selection of an item on one list display can then automatically trigger through DOCUS the appearance of the associated or follower display.

- o Write a key program calling it MSRETR as shown in Slide 17. The references to console features, such as lights, can be accomplished by symbolic notation. Also displays are called symbolically from the Library. List and Format displays have well defined meanings and structure so that, for example, it is possible to move automatically through an entire tree structure of connected displays with simple programming logic.
- o Assign MSGETR to a specified Function Key.

Employing the DOCUS techniques shown above results in increased programming efficiencies by reducing implementation time by factors of 10 to 50.

5.0 SUMMARY

DOCUS represents a new software concept for facilitating the implementation of display systems. DOCUS is both an operating concept for on-line display console usage and a man/machine procedure implementation tool. Through its Display Oriented Language, powerful conversation mode programs can be constructed by pursuing the following steps:

SLIDE 18

- o Design of the total procedure by identifying all Function Keys.
- o Determining the inter-key logic.
- o Generating pre-stored displays.
- o Inserting the symbolically addressable displays into a library.
- o Writing the Key Programs using DOL.
- o Assigning Key Programs to Function Keys.

The initial DOCUS capability has been implemented at Rome Air Development Center on a CDC 1604B computer employing the Bunker Ramo 85 Display Console. The user oriented application procedures which have been implemented include:

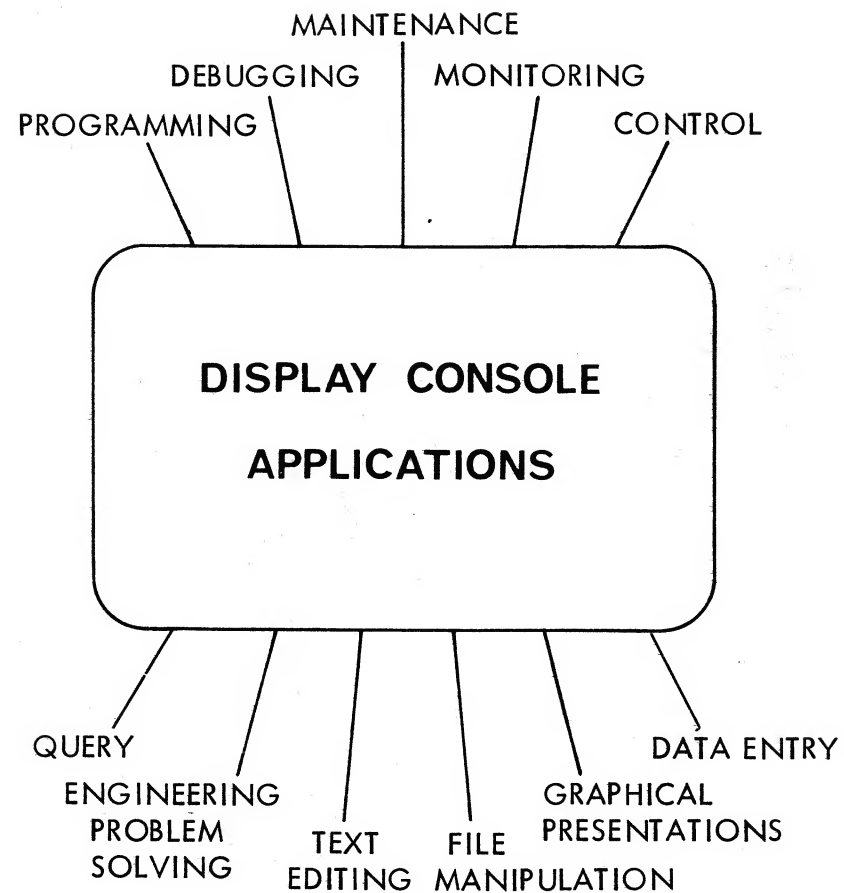
- o Console Programming in FORTRAN
- o Console Programming for CDC 1604 Assembler Language (CODAP)
- o Text Manipulating and Editing Capability
- o File Query and Management System
- o Operations for Display Library Handling

GENERAL PURPOSE SOFTWARE
FOR
DISPLAY SYSTEMS

THE
DISPLAY
ORIENTED
COMPUTER
USAGE
SYSTEM

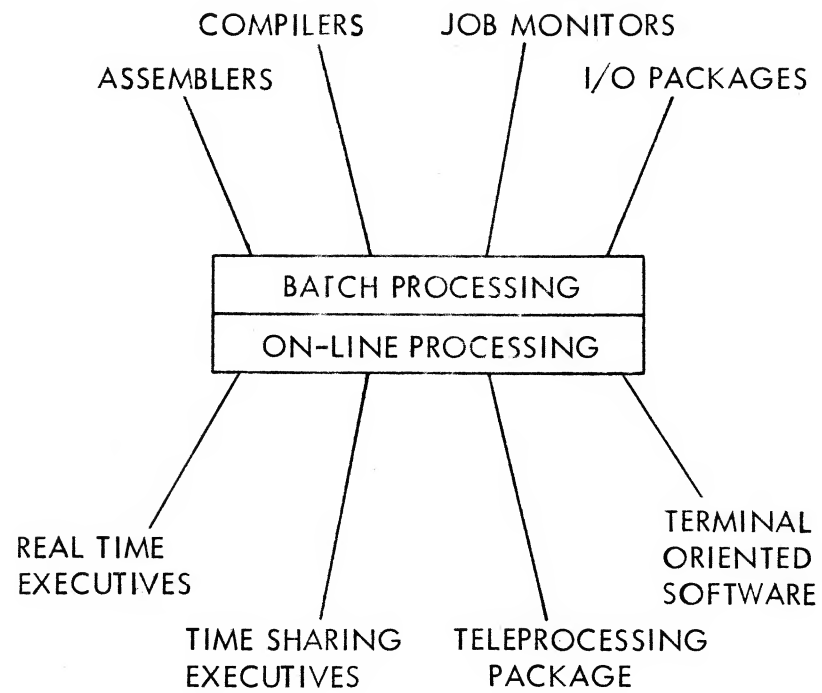


SLIDE I



SLIDE 2

DEVELOPMENT OF SOFTWARE TECHNOLOGY



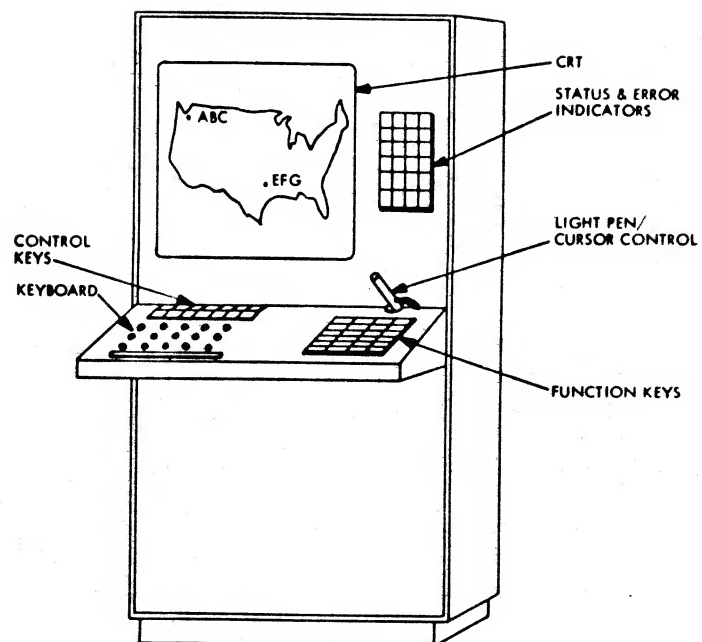
SLIDE 3



SLIDE 4

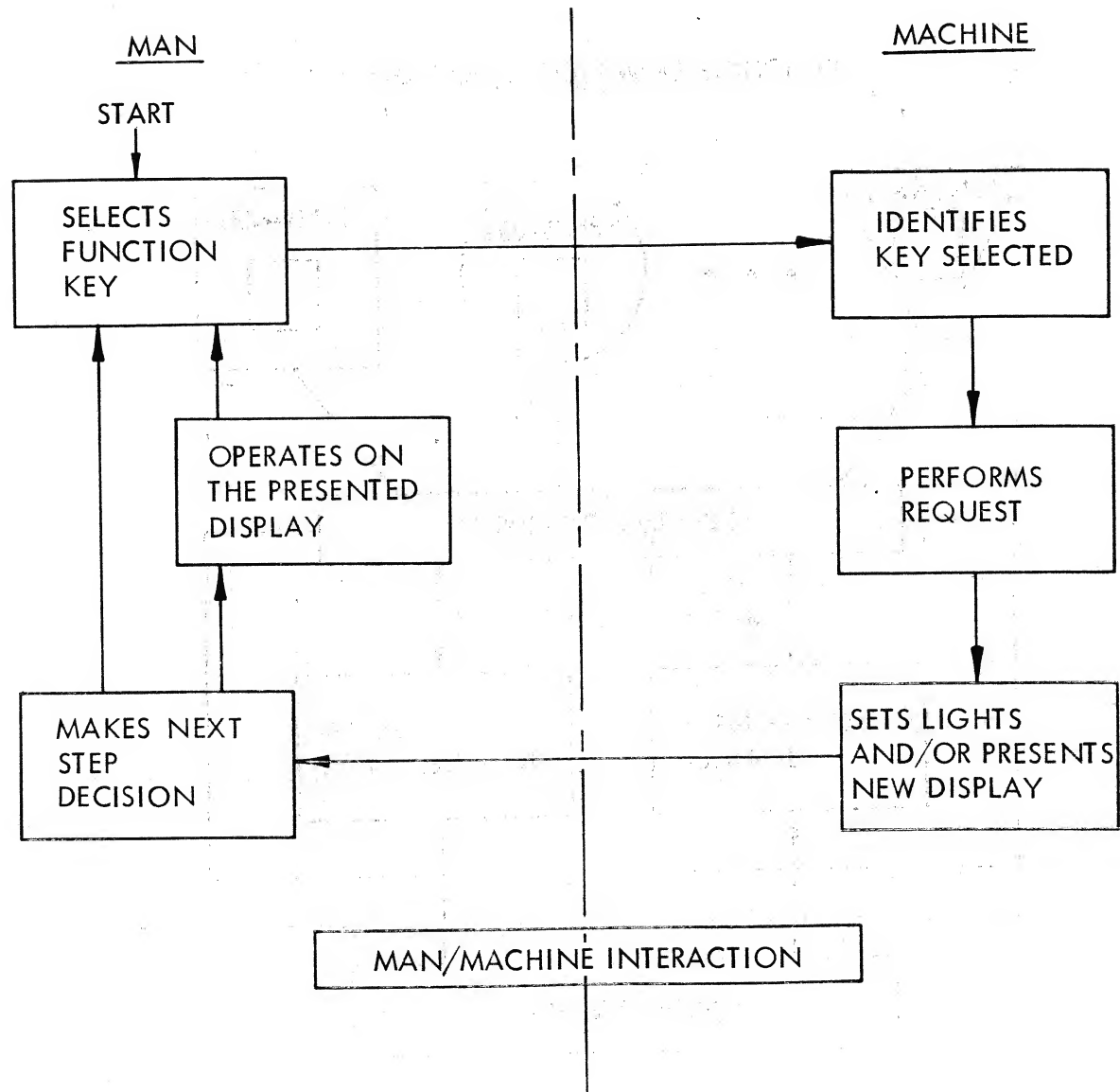


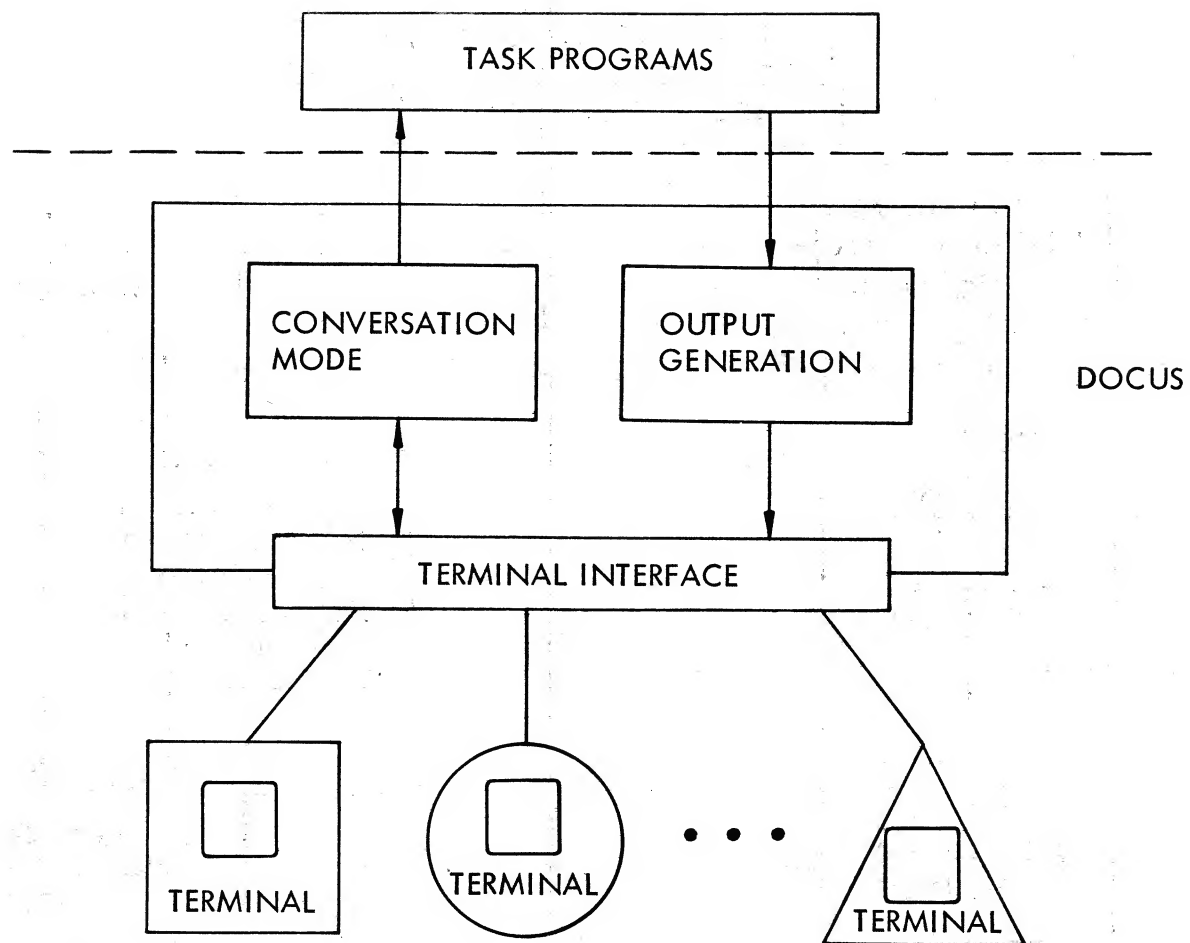
SLIDE 5



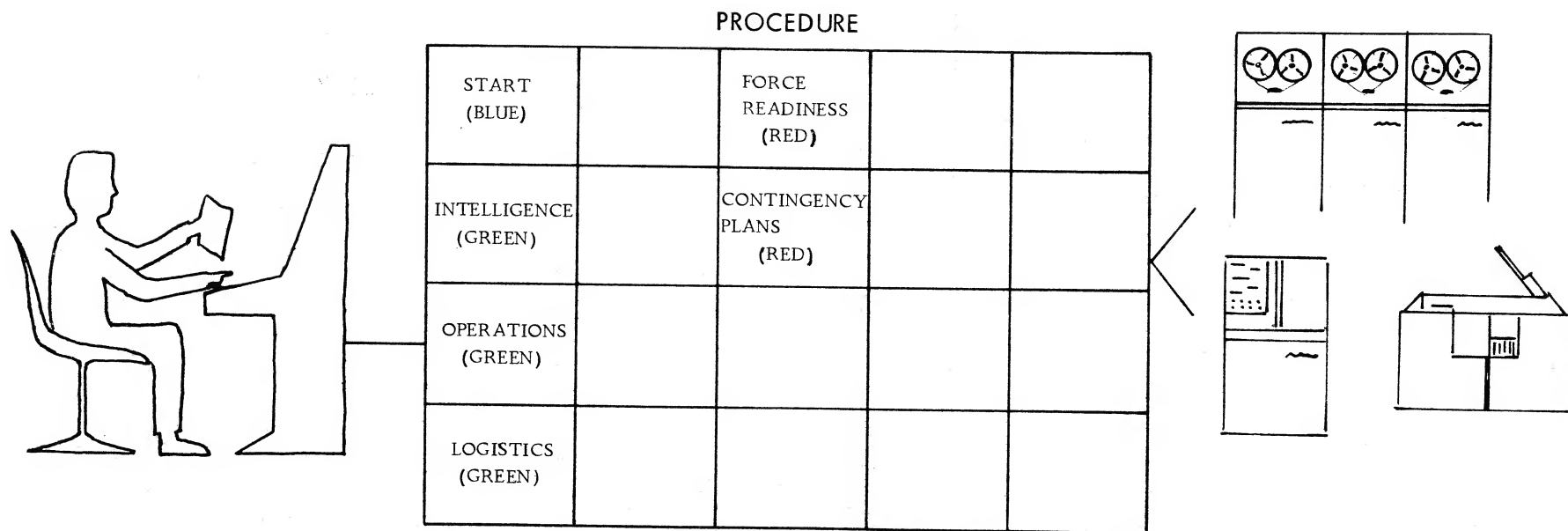
TYPICAL CRT DISPLAY CONSOLE

SLIDE 6





THE DOCUS SOFTWARE STRUCTURE



SLIDE 9

REDOPS MESSAGE ENTRY FORMAT

SECURITY CLASS: TS, S, C, U (LIGHT GUN CHOICE)

DTG: _ _ D _ _ _ Z _ _ M _ _ Y

UNIT NAME: _ _ _ _ _

UNIT ID CODE: _ _ _

UNIT DESIGNATOR: _ _ _ _

COORDINATES: _ _ ° _ ' _ " LAT
 _ _ ° _ ' _ " LONG

•
•
•

TELETYPE MESSAGE FORMATS

1. REDOPS
2. SITREP
3. COACT

•
•
•

USE LIGHT PEN TO SELECT REQUIRED FORMAT
 POINT AT NUMBER PRECEDING FORMAT

EXAMPLES OF DOL OPERATORS

1. CONTROLLING DEVICES ON CONSOLE

LIGHTS ON

LIGHTS OFF

ALARM ON

ALARM OFF

SAVE LIGHTS

RESTORE LIGHTS

NAME LIGHT

DEFINE LIGHT GROUP

TEST IF LIGHTS ON

READ CURSOR

PROCESS LIGHT PEN MESSAGE

2. COMPOSITION OF TEXTUAL DISPLAYS

DEFINE AREA

LITERAL AREA

DEFINE COMPOSITE

DEFINE STRING

LITERAL STRING

MOVE AREA

ROTATE AREA

FILL AREA

MOVE STRING

3. COMPOSITION OF GRAPHICAL DISPLAYS

DRAW LINE

MOVE POINT

ROTATE FIGURE

ERASE LINE

JOIN LINE

EXPAND FIGURE

CONTRACT FIGURE

COMPOSE PICTURE

DOCUS FEATURES

EXECUTIVE CONTROL

- MULTI CONSOLES
- MULTI PROCEDURES
- CONSOLE FEATURES

DISPLAY CONTROL

- DISPLAY CONSTRUCTION
- DISPLAY MANIPULATION
- LIBRARY HANDLER

CONVERSATION CONTROL

- PROCEDURE DEFINITION LANGUAGE
- PROCEDURE COMPILER
- OPERATING SYSTEM

EXAMPLE

MESSAGE DATA ENTRY

- TELETYPE TRAFFIC
- VOICE
- HARD COPY FORMATS
- TEXT

SLIDE 13

DISPLAY MSGSRC

MESSAGE SOURCES

- 1 TELETYPE
- 2 VOICE
- 3 HARD COPY FORMAT
- 4 FREE TEXT

USE LIGHT PEN TO SELECT MESSAGE SOURCE
POINT AT NUMBER PRECEDING TYPE

SLIDE 14

DISPLAY TTYMSG

TELETYPE MESSAGE FORMATS

1. REDOPS
2. SITREP
3. COACT



USE LIGHT PEN TO SELECT REQUIRED FORMAT
POINT AT NUMBER PRECEDING FORMAT

DISPLAY RDPFMT

REDOPS MESSAGE ENTRY FORMAT

SECURITY CLASS: TS, S, C, U (LIGHT GUN CHOICE)

DTG: _ _ D _ _ _ Z _ _ _ M _ _ Y

UNIT NAME: _ _ _ _ _

UNIT ID CODE: _ _ _

UNIT DESIGNATOR: _ _ _

COORDINATES: _ _ ° _ _ ' _ _ " _ LAT
_ _ ° _ _ ' _ _ " _ LONG

•
•
•

SAMPLE KEY PROGRAM

KEY PROGRAM: MSGETR
 LIGHTS ON (SI, R2)
 LIGHTS OFF (MSGETR)
 FETCH DISPLAY (MSGSRC)
10 WAIT FOR ENTRY
 IF LIST DISPLAY (20)
 IF FORMAT DISPLAY (30)
20 FETCH POINTED DISPLAY
 GO TO (10)
30 (FOR THIS EXAMPLE DATA ENTERED)
 (IN SELECTED TELETYPE MESSAGE)
 (FORMAT IS PROCESSED)
 .
 .
 .
 LIGHTS ON (MSGETR, MSGDLT)
 LIGHTS OFF (SI, R2)
 END

SLIDE 17

CONSTRUCTION OF CONVERSATION MODE PROGRAM

DESIGN TOTAL PROCEDURE

DETERMINE INTER KEY LOGIC

GENERATE PRESTORED DISPLAYS

INSERT DISPLAYS INTO LIBRARY

WRITE KEY PROGRAMS

ASSIGN KEY PROGRAMS

SLIDE 18

Design and Development of Software Packages

by RICHARD H. HILL
Vice President/Programming
Informatics Inc.

Reprinted with permission from American Management Association, Management Bulletin 79, published and copyrighted 1966.

informatics inc.



Main Offices:

5430 VAN NUYS BOULEVARD
SHERMAN OAKS, CALIFORNIA 91401

4720 MONTGOMERY LANE
BETHESDA, MARYLAND 20014

DESIGN AND DEVELOPMENT OF SOFTWARE PACKAGES

by RICHARD H. HILL
Vice President/Programming
Informatics Inc.

THE SOFTWARE INDUSTRY IS A YOUNG ONE, comparatively speaking. It is hard to know just where to date its beginning, but seven or eight years ago is reasonably close. Six or seven years ago the total market for the services of independent software companies was somewhere on the order of \$5 million a year. Currently, Informatics Inc. alone does business at that annual rate, and the total market for the software industry is probably somewhere between \$75

million and \$100 million a year. This estimate excludes the computer time that is sold by companies which are also in the software business. These figures are my credentials, as a representative of a software company, for discussing the design and development of software packages.

RICHARD H. HILL is Vice President—Programming, West Coast Office, Informatics Inc., Sherman Oaks, California.

At Informatics a software package is defined as "an integrated set of routines and associated documentation to accomplish practically anything that can be accomplished through EDP." Most of the work done by Informatics is of a type that can't be done through procedure-oriented or even problem-oriented languages. Our work is done usually in the language of the machine as represented by symbolic assembly programs, because we are usually called upon to implement systems that are a little bit out of the ordinary. Our specialty area is the field of on-line, real-time, man-machine systems, which are relatively complex. We design and build executive routines and monitor systems, display-programming systems, message-switching systems, and the like. Thus we think of software in terms of systems programs—that is, the programs that support the development of applications packages. However, I don't want to exclude any type of software package from consideration when discussing the problems of design, development, and—particularly—management of software activities. I think that the management problem especially is one that exists on all levels of planning and developing programs.

The basic concepts that were important in the development of modern computing systems are three. The first was the concept of symbolic language; it introduced the notion that one could express computer programs in a language other than that of the computer. This concept was first implemented by symbolic assembly programs, then by procedure-oriented languages, and today by some problem-oriented languages. The second basic concept was that of the generalized routine—a routine once written can be used over and over again as part of many software packages. Generalized routines were first used to do such things as work out square roots or sines and cosines. There are now a great number of generalized routines that do everything from those mathematical functions to sorting, report generation, and so on. The third important concept in the development of software packages was the notion of automatic transitioning from one activity to another, coupled with the idea of letting the computer do some of its own accounting work. From these basic concepts has grown the whole family of systems programming activi-

ties, including compilers, monitor systems, executive routines, generalized input-output systems, and the like. From a management viewpoint the batch-process monitor, in which jobs are introduced to the computing system one after another in a continuous stream, was a most significant forward step. The computer provides the transition from one activity to the next and does the necessary accounting as it goes. However, this is essentially a serial process in which the efficiency of the total computing system is limited by the interaction of the program with its data.

As the relative cost—that is, the cost-effectiveness—of hardware improved it became possible to utilize large segments of hardware for overhead functions and thus make feasible the timesharing of the equipment through multiprogramming. This step made it possible for jobs to be mixed and for the efficiencies of one job to be used to offset the inefficiencies of another. In the ideal situation a computer-bound program shares the computer with an input-output-bound program, and both of the programs together make maximum use of the available hardware.

The feasibility of these approaches really was not established until the current generation of computers, when electronics became so cheap and so reliable as to make feasible the dedication of a significant amount of hardware to the overhead functions necessary to allow such things as multiprogramming to take place.

Now there is a concomitant that must be accepted along with the great capability which is present in the third generation of computers. That concomitant is that all these overhead functions are accomplished by programs and, furthermore, these programs are among the most complex yet encountered. An example of this complexity is a program which has some 10,000 instructions in it, about 80 percent of which are yes-no decisions. The number of paths through that program is literally incalculable. This particular program is a systems program that is part of a large, modern computing system. It is something of a paradox that the hardware for this system is inexpensive and reliable—probably, the hardware is more reliable than the software. But we don't really have any good way of testing the software other than exercising it for a considerable length of time. In this respect, as in

many others, the hardware technology in computing is far ahead of software technology.

This problem of complexity in software to a great extent becomes the management problem of knowing where to cut off the development process. Cutting off the specification process before implementable limits are reached results in control. Anything less is tantamount to allowing invention to run wild.

This concept can be developed a little further. We have seen the use of the term "architecture" in connection with computer systems today. It is an apt term because it refers to the fact that software has developed beyond the point where the programmer is an artist. In effect, he has become an architect. The distinction is this: In the infancy of computing, a man approached a bare computer with an understanding of how the hardware worked and essentially created through his own artistry a program to solve a given problem. He may or may not have left traces other than the code behind him. His artistry was revealed in the code itself, and if it ran it was a wondrous thing. But the modern computing architect cannot afford this luxurious kind of creation. Rather, he has to view his activity as part of a complex, interacting system, just as a large, modern building is a complex, interacting system. The scope of his design activities has grown such that, although one person can possibly conceive the grand design for a large programming system, he cannot possibly hope to carry it out by himself down to the finest detail of design. So, like the architect who designs large, modern buildings, the architect of computing systems must communicate his grand design to his associates and coordinate their activities in its implementation.

This is easy to say, but it is a very difficult thing to do, because this sort of architecture is really in its infancy. Anybody who has been associated with the development of IBM System/360 would agree that the concept of architecture certainly exists there. The implementation has been a matter of infinite pain to a great many people. This is not to say that the implementation will not be good or that it will not be achieved, but the fact that programmers lived through something like that attests to the tenacity of the

human mind and its inexhaustible appetite for challenge.

There are many management problems associated with the successful implementation of a large-scale system. These problems are magnified by the fact that the art of programming really has not progressed to the point where it is able to cope with large-scale efforts on a readily achievable scale. The basic techniques—the tools of our profession—have not been developed to the point where we are able to say with certainty how much, for example, a given program will cost or how long it will take to write it, even after it has been described to us in great detail. Consequently, in our particular business there is a high element of risk involved in many types of activities. But it is the type of risk that we in the software industry are willing to accept on the basis of having had a considerable degree of experience, at least at a certain level. However, for people who do not have a similar reservoir of experience there are no tried and true methods to substitute for the experience. For example, there is no standard way of acquiring, or even stating the requirements of, a program in a sufficiently general form to be able to provide a general solution. We are reduced to the technique of analysis which consists of asking people, "What is it you want?" This question may have to be asked more than once because very frequently people don't know what they want or don't know how to state it. It is not infrequent for people to make changes as the program develops and even after it is completed. This is normal; this is why program maintainers exist. But it does point out the problem of finding a language which is sufficiently general to state requirements and the problem of finding programming techniques that are sufficiently flexible to permit the inevitable changes to be made at minimum cost and minimum risk of introducing new errors.

At Informatics Inc. we identify in the programming process some six or seven basic functions. First, when we attack a problem we prepare what we call a functional specification, which is the initial statement of requirements that we give to a customer and ask that he approve as the basic working document for further

design activities. Once we have agreed on a functional specification, which is effectively a statement of requirements, we produce a program design specification in which not only the externals of a program but also the internal make-up of the program, its various constraints, its reactions with other programs, and so forth are described. When this document is completed to our satisfaction and the customer's, we enter a coding phase. During this coding and the subsequent check-out of the program, we will be doing final documentation. And usually after check-out there will be some additional documentation to complete. Finally we will install the program—that is, bring it to a customer's location, make sure it runs on his computer, and train his people in its use and maintenance.

Note that only two of these six steps are directly concerned with the machine—the coding and check-out phases. The others are concerned with thinking and paperwork. If there were a less traumatic way to prepare documentation than by writing it down we probably would use it. But the fact remains that, in terms of today's techniques, the best way we have of capturing the ideas that go into a program and describing what a program does is by writing it down.

Documentation is all important because we don't document just for ourselves but primarily for other people. And the impression that we make has to be a favorable, lasting one or else we don't get any repeat business. Documentation thus is an absolutely essential investment from a management point of view because, unless the documentation comes with the program, the money spent on the program is in effect wasted, whether it was an in-house development or an outside purchase.

The first law of programming management is not to let programmers make decisions—at least, not until they become something more than programmers. All too often in the past, decisions have been made in the computing shop by default.

To leave important decisions, such as what your programming languages should be, to a person who has naturally a biased interest in the decision—the programmer himself—is to defeat

the effectiveness of the program as well as of the program manager. Programmers don't usually think like managers. They think like people, people who want to take the least painful route to accomplishing a set task and get it over with as soon as possible. These goals unfortunately are inconsonant with management's goals. The first problem of a manager of programmers, no matter what level he is operating on, is to understand that he has to impose some degree of management discipline upon his programming activities or else the whole thing is going to fail.

Programming is a manageable activity, but it takes a high degree of skill and knowledge of programming in order to manage it properly. Programming activities should be managed much like any other engineering or development activity, and it is very important to make preliminary estimates and budgets and to develop detailed schedules. In fact, Informatics uses the PERT technique when its projects are big enough. There should be a rigid and continuing system of monitoring the programming process, which in turn implies that functions in programming situations must be monitorable. In other words, you must establish milestones so that a check can be made to see whether or not the milestones have been met.

It is evident, then, that the important thing for management to understand about software development is the programming process. Reasonable schedules and reasonable milestones cannot be established unless management understands what the programmer has to go through to accomplish his work. One of the problems encountered repeatedly in selling software to private industry is lack of realism in looking at the software development process. This is found less often in selling software to government, where military and NASA have provided a pretty large background experience. But all too frequently, people in private industry expect the specification model to be absolutely correct. Of course, it isn't. Neither are the programs the first time around.

In addition, management must realistically assess the capabilities of its staff—in other words, be sure what its programmers can do. Of the

some 100,000 people in the United States today who are writing instructions for machines, there are perhaps only about 10,000 who are really professional programmers. Finally, sound ways to measure programming productivity must be developed. One of the reasons the whole profession has fallen behind in establishing measures is that the types of programming activities haven't been defined well enough so that we can measure productivity. Measures such as two instructions per hour, 40 instructions per day, and 60 instructions per day are realistic for certain types of programming activity. Yet more complex jobs may be estimated at one instruction per hour, ten instructions per day—and even these estimates may not be conservative enough. The difference, of course, is in the relative complexity of the jobs undertaken.

The amount of documentation also varies from job to job. Some programs may demand

one page of documentation for every 40 instructions in the finished program, in addition to the comments and the listings. Thus, for a system we recently produced that had about 45,000 instructions, there were 1,000 pages of documentation. The documentation included program descriptions, flow charts, write-ups, functional specifications, and so forth. This is a considerably higher level of documentation than most programs produce or even need; for example, a sort run on a well-defined master file certainly doesn't require a high level of documentation, even though there may be far more instructions involved.

When practical measures are established and related to problem complexity, management will be able to use these tools to establish more effective control of software. Efforts of independent software organizations and talented individuals should be directed toward this goal.



TIME-SHARING AND MULTIPROCESSING TERMINOLOGY

by ROBERT A. COLILLA
Informatics Inc.

informatics inc.



Main Offices:

5430 VAN NUYS BOULEVARD
SHERMAN OAKS, CALIFORNIA 91401

4720 MONTGOMERY LANE
BETHESDA, MARYLAND 20014

TIME-SHARING and MULTIPROCESSING TERMINOLOGY

toward standardized usage

by ROBERT A. COLILLA

In the general areas of time-sharing, multiprogramming, and multiprocessing, some terms in current use are very imprecise. It is hoped that as a result of this paper, some of the meanings will be clarified and, perhaps, a step gained in the direction of standardized usage.

When one thinks of time-sharing, some well-known systems immediately come to mind. Among these are MAC at MIT, the SDC Command Research Laboratory system, the ATLAS time-sharing system at Cambridge, and the RAND JOSS system. The MAC and SDC systems are similar in overall approach, but JOSS and the ATLAS system differ enough from them and from each other to merit special attention. All four systems are characterized generally by the fact that the central processor is not required to complete one job before starting another. Three of the above systems have an on-line, perhaps real-time, character in that communication between users and machines is made through on-line consoles. Since this is significant to the notion of time-sharing, the character of on-line systems is discussed first.

In an on-line system, a man or some device is either supplying information to a computer and/or waiting for the computer to supply him or it with information. There is always the implication that time constraints are important in the sense that either communication must occur within very precise intervals or, at least, that frequently delayed responses will defeat the purpose of the system. One also speaks of peripheral devices as being on-line to a computer if there are channels connecting the peripherals to the computer. Clearly the existence of a computer with an on-line device, alone, is not sufficient to make an on-line system. If such were the case, what system would not be on-line?

The characteristic of systems that really differentiates between those that are on-line and those that are not is the extent to which the computer may regulate the rate at which it accepts input data and transmits output data, and the extent to which lack of regulatory control influences the design of the system. In general, the less regulatory control a computer has over input/output rates and the more influence this lack of control has on the system design, the more on-line a system becomes.

In particular, when a system is such that the computer has practically no control over its I/O rates and where system design is entirely oriented around ensuring that the computer is ready to receive and transmit in time, then that system is usually called real-time. Real-time systems are, therefore, those on-line systems that manifest the extreme aspects of on-lineness.

It should be clear from the preceding that neither on-line nor real-time necessarily suggests any of the salient

features of time-sharing. On-line systems require neither many users nor human users to be qualified as on-line. As was stated earlier, however, some time-sharing systems do have an on-line character. Many users supply information to a computer, and it is required that a time-sharing system be designed to respond to the many users at frequent intervals. To get a better look at the term time-sharing and its companion term multiprogramming, it is good to recall some recent computer history.

time-sharing and multiprogramming

With the advent of asynchronous input/output operations in the late fifties, it became possible to perform simultaneous operation of a computer's central processor and its I/O processors. Programmers immediately set themselves to the task of using this new hardware capability to maximum advantage. Initially this meant better organization of individual programs. Input/output operations were strategically placed to achieve maximum use of the central processor. It did not take long, however, for the idea to be extended to that of operating two programs "simultaneously" so that one performed input/output operations while the other used the central processor. The ATLAS scheduling system carried the idea to its greatest extent. In ATLAS, all jobs are fed into the system as soon as they appear. A scheduling program selects those jobs having input/output characteristics which will tend to put as many of the computer components into motion as is possible. The objective of the ATLAS scheduling system is to "maintain the fullest possible useful activity in those parts of the computing system which can function simultaneously; that is, to reduce to a



A member of the senior staff of Informatics Inc., Bethesda, Md., Mr. Colilla is currently designing portions of the INTIP system of the Rome Air Development Center, Rome, N. Y. He was previously at RCA, where he designed portions of Project ACSI-MATIC for the Defense Intelligence Agency. He holds a BA in mathematics from New York Univ.

minimum periods of idleness in any part of the system which is required for further use."¹

Since, in the implementation, the central processor is required to transfer control frequently from one program to another without necessarily waiting for any of the programs to terminate, the terms multiprogramming and parallel programming arose. The latter is rarely used anymore. Also since portions of programs are sharing the central processor sequentially in time with all the other program portions, the term time-sharing arose.

As the above techniques were being developed, people were experimenting with the idea of connecting many electric typewriters to a computer and using them as on-line I/O devices. It was recognized at the outset that if men were going to use these typewriters as on-line devices, there would be a lot of very slow input/output operations. Not only would programs be delayed because of the slowness of the transfer rates of these devices, but they would be further delayed by the users' far slower rates of typing. Time-sharing is the logical technique to employ for this situation.

Time-sharing, however, is now embedded in an on-line environment and, as in all on-line environments, there are special time constraints imposed on the design. Whereas previously it was only necessary to switch from program to program to maximize utilization of all the components of the hardware, in the on-line environment there is the additional requirement to pass control to the jobs of the different users at frequent intervals so as not to ignore any user for any substantial length of time lest he become disenchanted with the system. With this additional demand, even if a single job program were exercising the components of the computer to their greatest extent with maximum efficiency, that program would have to be suspended periodically in order to make the system available to the other users.

This additional aspect to the notion of time-sharing is quite significant. It is so significant, in fact, that the requirement to respond to all users of the system has become the major characteristic of time-sharing systems and the term has become identified with this characteristic almost to the exclusion of the previous notion—that of maximizing utilization of all the components of the hardware. The term time-sharing, therefore, is used in both of these two senses.

The meaning of multiprogramming has also changed somewhat. This change is a direct result of the changing meaning of time-sharing. Whereas previously one might have described multiprogramming as the operation of a central processor that executes a number of programs in fractured fashion for the purpose of maximizing the use of the components of hardware, it would now be described without the last motive-supplying phrase. That is, the reason for performing the fractured operation is no longer part of the description.

multiprogramming and multiprocessing

As is usual in developing a concept, it is as important to say what the concept is not, as it is to say what it is. The term multiprogramming is often coupled with the term multiprocessing, with the implication that the two terms mean related but different things. It should be stated first that the processing part of multiprocessing refers to processor and not to process. Multiprocessing suggests the simultaneous operation of a number of processors. In distinction, multiprogramming is confined to the

operation of a single processor. In fact, it makes much more sense to talk about a multiprogrammed processor than it does to talk about a multiprogrammed computer or system.

The notion of a processor is basic enough to multiprogramming and multiprocessing to merit special attention. Despite any realization in hardware, a processor is conceptually a device that operates serially. It is this character of a processor that gives birth to the notion of multiprogramming. Multiprogramming can be said to be that operation of a serial processor which permits the execution of a number of programs in such a way that none of the programs need be completed before another is started or continued.

As was said before, multiprocessing implies more than one processor; yet it has to imply more than just this for otherwise any group of simultaneously operating computers would qualify as a multiprocessing system. Bright² has added the requirement that all the processors must have a common, jointly-addressable memory. It also seems necessary to require that no processor be dependent on another processor in order to operate. Another distinction can be noted here. Multiprogramming is, at least at present, a software notion. Multiprocessing, on the other hand, so far is exclusively a hardware notion. In searching for a possible software meaning for multiprocessing, one finds the following: Suppose one has a single job and a number of processors. Suppose, further, that the job can be divided into a number of parts, some of which can or have to be performed simultaneously on different processors, but some of which also must wait for the completion of other parts before they can be executed. Multiprocessing can be defined to be the ability to execute this divided job successfully.

To look at this in a slightly different way, multiprogramming can be said to be the task of fitting a single serial processor for many jobs, and multiprocessing is the task of fitting many serial processors for a single job. Such a definition, however, is more than most people have in mind when speaking of multiprocessing. This software notion could be called divided job processing. It may be added that divided job processing does not really require the involvement of more than one processor to have meaning. There are people who indeed define multiprocessing as being divided job processing irrespective of the number of processors involved. At this time, however, it is perhaps best to leave multiprocessing a hardware notion and wait to see if any software notion develops.

extended notions

It was said earlier that a processor is a device that performs instructions serially. Although it was implied that the instructions of concern were machine instructions, this does not have to be the case exclusively. One may think in terms of an extended machine just as well. If, for example, one thinks in terms of FORTRAN instructions, one can define a FORTRAN processor as a device for performing FORTRAN instructions serially. One can then multiprogram this processor by fitting it for a number of FORTRAN jobs. Similarly one may think in terms of many FORTRAN processors operating simultaneously to achieve a multiprocessing system.

The joss system is a time-sharing system which is entirely interpretive. The job programs never really have control of the central processor. One cannot say that the central processor is multiprogrammed if one thinks in terms of machine instructions. If, however, one thinks in terms of joss instructions, one can say that joss is a multiprogrammed joss processor.

¹ Howarth, D. J., and others, "The ATLAS Scheduling System," *Proceedings, 1963 Spring Joint Computer Conference*, p. 60.

² Bright, Herbert S., "A Philco Multiprocessing System," draft preprint, *Proceedings, 1964 Fall Joint Computer Conference*, p. 1.

There are two other notions that should be mentioned. In the definition of multiprogramming it was said that the completion of a job is not required for the continuation of another. Nothing was said—nor should anything be said—of how it is decided when control should be passed from program to program nor of how it is decided what the succession of programs should be. That these decisions have to be made by all multiprogramming systems is not being denied here. Rather, it is being emphasized that there are many different ways for making the decisions and that no one way should be strongly associated with the term multiprogramming to the exclusion of all others.

In ATLAS, program switching occurs when the program being executed requests an input/output operation. In both MAC and the SDC Command Research Laboratory, transfer may occur either when input/output operations are requested or when a prescribed amount of time has elapsed. The amount of time allotted to each program is a multiple of a basic unit of time, called the time quantum. In joss, an interpretive system, switching may occur after the execution of each statement written by the user. As far as the author can tell, no name has been assigned to the task of determining when switching should occur among programs. Program suspending is suggested.

The second item, that of determining what the succession of programs should be, is scheduling. Like program suspending, scheduling is an intrinsic part of all multiprogramming systems, but is implemented differently in different systems. In some systems, there is a single queue. Once a program is suspended, it is placed at the bottom of the queue. The next program to be executed is taken from the top of the queue. Other systems maintain two or more queues where each program is assigned to a specific queue on the basis of externally-assigned priorities, amount of storage required, availability of peripheral devices, various combinations of these, and many other program characteristics.

In light of the preceding, what can be said about time-sharing? It seems best to follow the trend and let time-sharing take on its newer sense. One could say that time-sharing is the use of a multiprogrammed processor for the purpose of permitting a number of users to operate the processor in such a way that each is unaware of the use of the processor by the others. It may turn out, however, that as sophistication increases multiprocessing will become involved in time-sharing. It is perhaps best, therefore, to leave out the multiprogrammed part. Time-sharing can then be said to be that operation of a computer facility that permits many users to operate the facility simultaneously or apparently simultaneously in such a way that each is or can be completely unaware of the use of the facility by others.

For those who are interested in the maximum utilization of all the components of the hardware, one can still use the term multiprogramming, but has to add some motivating phrase. Perhaps component utilization multiprogramming will suffice. For those who want to talk about the multiprogramming involved in time-sharing systems, perhaps user multiprogramming will do.

epilogue

We are an industry that has an abundance of vocabulary. For a group interested in advancing the art of communication between men and machines and for one concerned with the accurate retrieval of information, we have not performed well in communicating among ourselves. Indeed we almost seem to have invested more effort in developing catchy, semi-absurd names than we have in developing clear, meaningful ones. It is earnestly urged that we begin at this time to approach our naming with as much a scientific purpose as we do our programming

and building of computers. In this way we will develop a healthy respect for words. We will choose them more carefully and, as a result, will communicate faster and with much less effort than is possible at present. ■

SUMMARY OF DEFINITIONS

Processor:	A device capable of performing a set of instructions serially.
Multiprogramming:	That operation of a (serial) processor which permits the execution of a number of programs in such a way that none of the programs need be completed before another is started or continued.
Multiprocessing:	A computer configuration consisting of more than one independently initiable processor, each having access to a common, jointly-addressable memory.
Job Dividing:	The task of dividing a job into a number of parts, some of which can or have to be performed simultaneously, but some of which require the completion of other parts before they can be executed.
Divided Job Processing:	The task of programming one or more processors to permit the execution of divided jobs.
Program Suspending:	The task of determining when transfer of control should take place in a multiprogramming system.
Scheduling:	The task of determining what the succession of programs should be in a multiprogramming system.
Time-Sharing:	The operation of a computer facility that permits many users to operate the facility simultaneously or apparently simultaneously in such a way that each is or can be completely unaware of the use of the facility by others.
On-Line:	The characteristic of a system whereby input/output rates are not under complete control of the system and where this lack of control significantly influences the design of the system.
Real-Time:	The characteristic of a system in which there is so little control of input/output rates that the entire design of the system is dominated by the need for insuring the system can receive and transmit at rates commensurate with the input/output rates.

Display Software Technology

By

George Collins
INFORMATICS INC.

Reprinted with permission from SIGNAL, July 1966
Published and copyrighted 1966 by the Armed
Forces Communications and Electronics Association

informatics inc.



Main Offices:

5430 VAN NUYS BOULEVARD
SHERMAN OAKS / CALIFORNIA 91401

4720 MONTGOMERY LANE
BETHESDA, MARYLAND 20014

DISPLAY SOFTWARE TECHNOLOGY

GEORGE COLLINS, INFORMATICS, INC.

WITH THE advent of asynchronous input/output operations in the late fifties, it became possible to perform simultaneous operations of a computer's central processor and its input/output processors. Since that time many ideas have been advanced and experimentally tested aimed at a better utilization of computing systems. At first the aim was merely to phase the input/output subtasks of a given problem in a manner such that computational processing and input/output operations were done in parallel. For most problems, however, even optimal phasing could not fully utilize all components of a computer system. The technique was then enlarged allowing a computer to work on more than one problem simultaneously, performing input/output operations for one problem while processing information for another. From these experiments arose the concept of time-sharing a computer among several users. This, in turn, caused the desired goal of better utilization of computing systems to be broadened further to include not only the more efficient usage of all components of a computing system but also an improvement in the communication and interaction between man and machine.

Thus, the trend which had developed during the first decade of computer usage to isolate man with his slow response time from the machine operating in microseconds was

suddenly reversed. By allowing one computer to service many users on a concurrent basis, the mismatch in response times between man and machine is no longer an economic barrier to letting a user interact directly with the computer.

Many problems, which previously could not be solved on the one hand by man alone because of the large amounts of data to be processed or on the other hand by machine alone because of the requirements for human judgment and insight in frequent redirection of the solution algorithm, now appear to be feasible and economical for a man/machine team.

As a result, interest in on-line interactive systems has increased rapidly during the last few years. Much research and development is occurring in the design and organization of multiprocessing/multiprogramming hardware and software. A part of this effort has been spent in the development of the consoles or terminal devices to be used in communication between man and machine. Cathode ray tube display consoles have been developed with a wide range of capabilities and features and with a corresponding wide range of costs.

Display consoles are particularly advantageous as a terminal device in an on-line interactive system for several reasons. Display consoles are fast, with transfer rates of tens of thousands of characters per second contrasted with the transfer rates of tens of characters per second of typewriter consoles. Display consoles are quiet, eliminating the distracting effect of the clatter of a computer driven typewriter. Dis-

play consoles are flexible. For example, it is relatively easy to insert material into the middle of a display or to close up lines or to reformat in any desired way. In addition, information transmitted between man and machine may be graphical as well as textual.

These advantages, of course, must be weighed against the disadvantages. Display consoles are today still relatively expensive. Hard copy of a display output requires additional hardware devices and thus may be costly and difficult to obtain and display consoles, because of their flexibility, require more elaborate software.

Extended Machine Concept

The terms, man-machine system, and man-machine communication, are frequently used in describing and characterizing on-line computer systems. It should be noted that, in this context, the "machine" is not merely computer hardware. What is meant, instead, is an "extended machine" comprised of electronic computer hardware components augmented by a collection of computer programs. The user of an on-line system may not and, indeed, need not be aware of the existence of the system's software programs but it is this software which controls and directs the hardware in responding to the requests and actions of the user.

The software of an on-line interactive system serves many purposes. It may be dedicated and oriented to a specific application or problem area. In this role, it provides the appropriate responses to requests or actions of a user, performing the functions and calculations related to

the specific application. It may provide the tools and building blocks for constructing additional software. In this role, it does not directly support any particular application but instead provides the means for adding capabilities to the system for new applications. Or, software may serve in a utility role providing general functions, such as message formatting or input/output transmission, which are required by the applications and system building software.

During the last few years, many groups have worked and experimented with this new device, the display console. I can only mention a few of the more important projects highlighting the contributions made by each.

The SAGE System

Beginning with the early pioneering work on the SAGE system, display consoles have been an integral part of military command and control systems. Most of these consoles have been special purpose developments for a particular system. The usage of the consoles has been primarily for the geographic presentation of military situations or status of forces and for short alphanumeric messages in response to queries initiated at the console.

The work done by Ivan Sutherland at Lincoln Laboratory on SKETCHPAD was a major step in the use of display consoles for the generation, composition, and manipulation of graphical displays. SKETCHPAD was implemented initially on the TX-2 computer which included among its peripheral components two CRT scopes and related devices. SKETCHPAD allowed a user to draw lines and compose pictures, to move or copy portions of the pictures, to impose constraints upon picture parts (such as, lines must remain parallel during any movement), and to file and retrieve pictures and picture parts from a library.

At the Bell Telephone Laboratories, Kenneth Knowlton and his associates added the dimension of time to displays and created animated movies using a computer/display console configuration. The movies were produced by creating displays on a Stromberg Carlson

4020 console representing the position and attitude of a tumbling satellite at discrete time intervals, photographing the pictures displayed one frame at a time, and then projecting the result as a movie. Control parameters input to the computer program could be changed and the resulting gyrations of the simulated satellite could be observed visually via the movies produced.

The team of Glenn Culler and Bernard Fried at TRW/Space Technology Laboratories used a computer/display console configuration to bring mathematicians and physicists into close interaction with a computer in determining and carrying out solutions to problems in mathematical analysis. The analyst has available a large repertoire of basic functions which he can combine to form more complex functions. At any point, he may view a graphical display of intermediate results. Observing visually and directly the evolution of his solution scheme frequently provides an analyst with the insight needed to redirect his attack on a problem.

At the General Motors Research Laboratories, an integrated approach has been made in assisting automotive design engineers. Design Augmented by Computers (DAC) was initially implemented on an IBM 7094 computer coupled to a special image processing system designed and built especially for this effort. A design engineer can create drawings and experiment with these, manipulating and reworking the drawings as desired. A finished drawing can then be converted to hard copy form. The designer can overlay two pictures for comparison purposes. Non-graphic communication and processing is also provided.

While display consoles have been used frequently in information storage and retrieval systems, their usage has been mostly limited to the display of the answer to a query. The user initiating the query was assumed to know the details of the structure, format, and even the content of the data base with which he was working. AESOP, developed at the Mitre Corporation, permits a user who is not well acquainted with the data base to browse through it, guided continuously on the choices of actions available to him. When interesting data is found, a user may establish temporary working copies

of it, which he may then manipulate as he desires. Hard copy reports in user designed formats may be obtained.

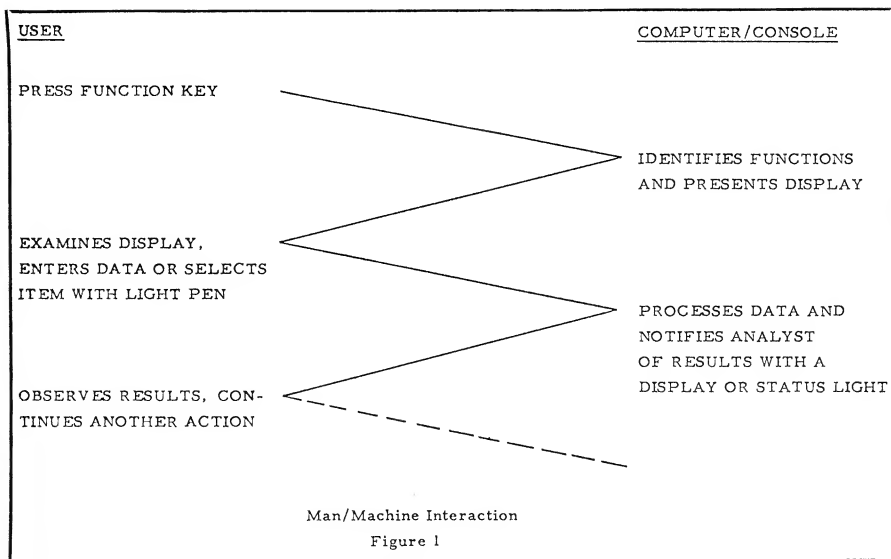
All of these systems and most of the other experiments and studies investigating the usage of display consoles have been special purpose developments. The software associated with these systems is dedicated either to a specific application or is highly oriented to a particular hardware configuration. This is typical of the history of data processing. First comes an experimentation phase to determine the utility and capabilities of a new hardware device. Then, general purpose software is developed. We have seen the evolution of assemblers, compilers, job monitors and operating systems in support of conventional batch processing operation of a computer. We are now at a point where display consoles have had their utility and capability demonstrated. Development of general purpose software is needed now to facilitate the usage of these devices.

Concepts and Philosophy of DOCUS

I would like to discuss briefly the Display Oriented Computer Usage System (DOCUS) which is currently being developed by Informatics for the Rome Air Development Center. DOCUS is one of the several efforts currently underway to provide the general purpose software needed to effectively utilize CRT consoles.

Under the Display Oriented Computer Usage System (DOCUS) effort, techniques are being developed to facilitate the implementation of on-line procedures and the operation of such procedures through a man/computer conversation mode. The users' areas of interest may be related to information retrieval, scientific research, command and control or business management. It is desirable to make these on-line users less dependent upon the professional programmer by simplifying the implementation and reducing the time for programming new applications.

The basic objectives of DOCUS are to design new techniques of man/machine communications; improve existing man/machine techniques; develop display console utilization methods; and implement and



test selected man/machine techniques.

The Display Oriented Computer Usage System is a fundamental software system responding to these objectives by providing on-line programming and debugging through the use of higher order languages; on-line development of sophisticated processing functions from simpler functions; and display oriented software tools to assist implementors.

A prototype of DOCUS is being implemented for a Control Data Corporation 1604B computer to which is attached a Bunker Ramo 85 (BR85) display console.

In the DOCUS concept, this man/machine communication is accomplished by a series of related steps that constitute a conversational mode of operation. First, the user performs an action (pressing a function key on the console, using a light pen, or entering data via a keyboard); the computer responds to the user's actions and presents to him a display or turns on or off some indicator lights; the user examines the computer's message and performs another action. The process continues until the user is satisfied or the computer is unable to respond to the user in a satisfactory manner and an alarm condition may occur. This operation is diagrammed in Figure 1.

The techniques developed in DOCUS are independent of time sharing, multi-programming or multi-processing. It is only when the economics of a system are considered or console-to-console interactions are considered that these modes of operation become impor-

tant. Since there is much work in progress in these areas, the implementors of DOCUS have borrowed freely from the variety of time sharing and multi-programming methods and concentrated on man/machine aspects.

Display Oriented Language

It was recognized at the inception of the project that an appropriate language was needed for efficient implementation of the software needed for the man/machine dialogue. A Display Oriented Language (DOL) is currently under development to satisfy this need.

The objectives and goals of the development effort on the Display Oriented Language are: to provide a suitable language for stating the display oriented functions of the required software; to be as equipment independent as possible in concept, design, and implementation; to allow intermixing of other languages (to whatever extent feasible and desirable) for stating the data processing and computational functions of the required software; to be of assistance not only in developing applications software but in developing system building and utility software as well; to provide a collection of general purpose utility programs for manipulating display consoles; to provide a translator (compiler, macro-assembler, preprocessor, or whatever) to transform DOL statements into machine executable code.

In order to achieve as large a degree of equipment independence as possible, it was necessary to examine the types of basic functional actions which can be performed by

man and machine without regard to any specific computer, display console, or applications area.

We find that a console user can perform one or more of the following functions in communicating with the machine: *Interrupt*—the user can take some action to interrupt the normal sequencing of the computer or to set "service needed" flag signalling that the console has information for the computer; *Select*—the user can make a selection either by activating a console selector device or by designating an element of the display on the CRT scope; *Enter Message*—the user can enter messages for transmission to the computer by an alphanumeric keyboard and other manual input devices.

The machine in turn communicates with the man by performing one or more of the following functions: *Set Indicators*—indicators are used to inform the user of the status of the problem, to guide him in the selection of the next problem step, or to inhibit interrupts (by hardware or software means); *Transmit Message to Display on Scope*—display message may consist of "blanks" to be filled in, selection list, informative message, or a combination of these.

Analysis of the above functions and of the hardware characteristics of various display consoles which are currently available led to a set of characteristics or hardware features of a generalized console.

A generalized display console will be assumed to have these features: *Indicators*—visual or aural devices having two states, "on" and "off"; *Selectors*—switch devices which when activated cause the device number to be registered and/or transmitted to the computer; *Attention Requests*—devices which can demand or request attention from the computer; *Attention Suppressors*—devices which either prevent attention requests from interrupting the computer or which indicate interruptions produced by attention requests are to be ignored; *CRT Scope*—with symbol display and with line vector capability; *Alphanumeric Keyboard*; *Pointers*—for designating a selected display element—a symbol or a line segment; *Various Manual Controls*—used to enter data (symbols or lines) on the scope and to manipulate (move,

copy, delete, and so forth) data elements on the scope.

A specific console of course, may have various quantities of these features. Certain features may be missing; certain features may be combined.

It may be of interest to measure the BR85 on which the prototype DOCUS is being implemented against this generic console.

The BR85 console considered from this generalized viewpoint has:

87 Indicators—60 function key lights, 25 status lights, a blinking error light, and an audible alarm.

62 Selectors—60 function keys, 2 overlays.

63 Attention requests—the 62 selectors above (note the combined functions in a single device) plus a light pen.

62 Attention suppressors—2 hardware interruption inhibitors (status lights 01 and 02) plus 60 function key lights which are interpreted by software to accept or ignore function key interruptions.

1 CRT scope—with 64 unique symbols and with line drawing capability.

1 Aphanumeric Keyboard.

2 Pointers—a light pen and a crosshair cursor.

There are also various manual controls which are not controllable by the computer.

DOL is designed to serve as a language for stating the display console oriented functions of a program. It will be compatible with FORTRAN which will be used for the purely data processing functions of a program. DOL statements may be separated into the following categories:

1. Declarations or Specifications—These are the statements which provide to the DOL compiler a description of the display console and its features. Included in this statement group are declaratives which allow a user to name a console device as he chooses and to refer subsequently to the device by this name.

Examples:

DEFINE SCOPE (CHARACTERS, 32, 64, RASTER, 384, 512)

NAME FUNCTION KEY (10, ENTER DATA)

DEFINE LIGHT GROUP (PAGELITES, 2, NEXT-PAGE, PREVPAGE)

2. Console Communications and

Control Statements—This class of statements is used to turn on or off various console devices, to interrogate console registers, to acquire console messages, and to transfer data between computer and console.

Examples:

TURN ON LIGHT (ENTER DATA)

WRITE CONSOLE (BUFFER1)

PROCESS LIGHTPEN MESSAGE

3. Textual Display Composition Statements—These statements are used to compose a symbolic or textual display. Included are commands for manipulating in various ways the two-dimensional strings of character making up a textual display. Statements are available to dissect user responses extracting message components from background information.

Examples:

FORM CHARACTER MATRIX FROM DISPLAY MESSAGE

DISSECT CHARACTER MATRIX

MOVE STRING TO CHARACTER MATRIX

4. Graphical Display Composition Statements—This group is used to generate and manipulate graphical displays composed of line segments.

Examples:

EXTEND LINE

ROTATE FIGURE

EXPAND FIGURE

JOIN LINES

5. System and Environment Linkages—These statements provide the linkages between a program and the executive control of DOCUS. They also provide facilities for storing displays in a library and for subsequently retrieving from the library.

Examples:

FETCH FROM LIBRARY (Display name)

WAIT FOR CONSOLE RESPONSE (ENTER DATA)

DOCUS Applications

As stated earlier, the objectives of DOCUS include not only the development of general purpose software for display consoles but also the testing of the adequacy of the software techniques by applying these to a wide range of problems. In these applications, the principal emphasis is on the establishment of methodology and development of tech-

niques for man/machine communication. Within the DOCUS project we are developing procedures in these diverse areas: on-line programming and checkout; file and data base management; text manipulation and editing; mathematical analysis and computation; and graphical plotting.

On-line programming via a display console will be possible in the prototype system using FORTRAN, DOL and CODAP (Machine Assembly language for 1604B). After initially defining a program, the procedure implementor will be able to make corrections, to compile, and to check out the program from the console.

Procedures are being provided for on-line file management allowing a user to define a file structure and format, to input data to update files, and to retrieve data from the files using a simple query language. Facilities will be available for specifying the content and format of reports to be displayed and then output via a hard copy device.

The text manipulation and editing procedures allow a user to input a stream of textual material, to cause the text to be displayed, rolling it backwards and forwards as needed to scan it. The user can insert, delete, replace, or copy text items ranging in size from a single character through words and sentences to paragraphs. The final edited text may be viewed and then output via a hard-copy device or written on an auxiliary storage device for future use.

Uses and Results

The mathematical computational procedures will provide the user of the display console with a full repertoire of arithmetic, trigonometric, matrix and statistical functions which he may use to operate on his data sets. Results of the operations will be displayed in convenient form.

Graphical plotting procedures will allow arrays of data to be processed and displayed in a variety of ways. Processing will include smoothing of data, curve fitting, data reduction and evaluation. Several graphical plots may be superimposed to allow easy comparison.

— — — — —

**THE DISPLAY ORIENTED
COMPUTER USAGE SYSTEM:
A GENERAL PURPOSE
SOFTWARE PACKAGE
FOR DISPLAY SYSTEMS**

informatics inc.



TIME-SHARING AND MULTIPROCESSING TERMINOLOGY

by ROBERT A. COLILLA
Informatics Inc.

informatics inc.



Main Offices:

5430 VAN NUYS BOULEVARD
SHERMAN OAKS, CALIFORNIA 91401

4720 MONTGOMERY LANE
BETHESDA, MARYLAND 20014

TIME-SHARING and MULTIPROCESSING TERMINOLOGY

toward standardized usage

by ROBERT A. COLILLA

 In the general areas of time-sharing, multiprogramming, and multiprocessing, some terms in current use are very imprecise. It is hoped that as a result of this paper, some of the meanings will be clarified and, perhaps, a step gained in the direction of standardized usage.

When one thinks of time-sharing, some well-known systems immediately come to mind. Among these are MAC at MIT, the SDC Command Research Laboratory system, the ATLAS time-sharing system at Cambridge, and the RAND JOSS system. The MAC and SDC systems are similar in overall approach, but JOSS and the ATLAS system differ enough from them and from each other to merit special attention. All four systems are characterized generally by the fact that the central processor is not required to complete one job before starting another. Three of the above systems have an on-line, perhaps real-time, character in that communication between users and machines is made through on-line consoles. Since this is significant to the notion of time-sharing, the character of on-line systems is discussed first.

In an on-line system, a man or some device is either supplying information to a computer and/or waiting for the computer to supply him or it with information. There is always the implication that time constraints are important in the sense that either communication must occur within very precise intervals or, at least, that frequently delayed responses will defeat the purpose of the system. One also speaks of peripheral devices as being on-line to a computer if there are channels connecting the peripherals to the computer. Clearly the existence of a computer with an on-line device, alone, is not sufficient to make an on-line system. If such were the case, what system would not be on-line?

The characteristic of systems that really differentiates between those that are on-line and those that are not is the extent to which the computer may regulate the rate at which it accepts input data and transmits output data, and the extent to which lack of regulatory control influences the design of the system. In general, the less regulatory control a computer has over input/output rates and the more influence this lack of control has on the system design, the more on-line a system becomes.

In particular, when a system is such that the computer has practically no control over its I/O rates and where system design is entirely oriented around ensuring that the computer is ready to receive and transmit in time, then that system is usually called real-time. Real-time systems are, therefore, those on-line systems that manifest the extreme aspects of on-lineness.

It should be clear from the preceding that neither on-line nor real-time necessarily suggests any of the salient

features of time-sharing. On-line systems require neither many users nor human users to be qualified as on-line. As was stated earlier, however, some time-sharing systems do have an on-line character. Many users supply information to a computer, and it is required that a time-sharing system be designed to respond to the many users at frequent intervals. To get a better look at the term time-sharing and its companion term multiprogramming, it is good to recall some recent computer history.

time-sharing and multiprogramming

With the advent of asynchronous input/output operations in the late fifties, it became possible to perform simultaneous operation of a computer's central processor and its I/O processors. Programmers immediately set themselves to the task of using this new hardware capability to maximum advantage. Initially this meant better organization of individual programs. Input/output operations were strategically placed to achieve maximum use of the central processor. It did not take long, however, for the idea to be extended to that of operating two programs "simultaneously" so that one performed input/output operations while the other used the central processor. The ATLAS scheduling system carried the idea to its greatest extent. In ATLAS, all jobs are fed into the system as soon as they appear. A scheduling program selects those jobs having input/output characteristics which will tend to put as many of the computer components into motion as is possible. The objective of the ATLAS scheduling system is to "maintain the fullest possible useful activity in those parts of the computing system which can function simultaneously; that is, to reduce to a



A member of the senior staff of Informatics Inc., Bethesda, Md., Mr. Colilla is currently designing portions of the INTIP system of the Rome Air Development Center, Rome, N. Y. He was previously at RCA, where he designed portions of Project ACSI-MATIC for the Defense Intelligence Agency. He holds a BA in mathematics from New York Univ.